CSE 414 Database Project Report

Yakup Talha Yolcu
1801042609

TRANSPORTATION AND ACCOMMODATION BOOKING SYSTEM

CONTENTS

CONTENTS **2**
PROBLEM DEFINITION **3**
TRANSPORTATION AND ACCOMMODATION BOOKING SYSTEM DESCRIPTION **3**
IMPLEMENTATION DETAILS **4**
USER REQUIREMENTS **4**
Person 4
Guide 4
Company 4
ER DIAGRAM **5**
TABLES **7**
SELECT EXAMPLE **9**
INSERT EXAMPLE - 1 **12**
INSERT EXAMPLE - 2 **13**
INSERT EXAMPLE - 3 **14**
UPDATE EXAMPLE - 1 **15**
UPDATE EXAMPLE - 2 **16**
UPDATE EXAMPLE - 3 **17**
DELETE EXAMPLE - 1 **18**
DELETE EXAMPLE - 2 **19**
DELETE EXAMPLE - 3 **20**
FUNCTIONAL DEPENDENCIES **21**
CHECKING NORMALIZATION **21**
VIEWS **23**
TRIGGERS **24**
Update person budget when accommodation is bought or canceled **24**
Update person budget when car rental is bought or canceled 25
Update person budget when a transport is bought or canceled 25
Can’t rent a car more than one week and check budget is enough 28
Cant accommodate on a hotel more than one month and check budget is enough 29
Check budget for transportation 30
JOINS **31**
RIGHT JOIN 31
LEFT JOIN 32
FULL JOIN 33
INNER JOIN 34
ATOMIC TRANSACTIONS **35**
While person adds a budget 35
While person removes a budget 36
While company transfers a guide 37
While company transfers a car_rental 38

PROBLEM DEFINITION

The main purpose of this project is manage the database of a travel agency efficiently. Agencies can manage tours, flights, bus trips, accommodations for persons, car rental for persons. With my experience , I did not see a system like that. This system combines all travel and accommodation occasions together. System will reduce the sightseers burden.

Companies will get benefit also, they do not need to track for each person. With tours, companies can combine flight or bus trips with an accommodation.

TRANSPORTATION AND ACCOMMODATION BOOKING SYSTEM DESCRIPTION

In this project, the database implementation and interface of a transportation and accommodation booking system is made. First of all, system can be used by 3 users.

- person
- guide
- company

Persons can be added by company with person_id, name, birthdate and a budget.
Guides can be also added by company. Guides are also persons. There is an inheritance. Guides can do whatever person do. In addition to that, guides can view assigned transportations them. Company offers these functionalities :

- plan bus trip
- plan flight trip
- plan accommodation on a hotel
- renting a car
- joining a tour (bus or flight + accommodation)

IMPLEMENTATION DETAILS

I used MYSQL for database and Java for development. I used Java Swing Library for pretty UI.

USER REQUIREMENTS

Person

Person can do :

- View / Buy / Cancel Car rentals
- View / Buy / Cancel Flight trip
- View / Buy / Cancel Bus trip
- View / Buy / Cancel Accommodation
- View tour
- Cancel tour joining

Guide

Guide can do :

- Do whatever person can do
- View guided transports

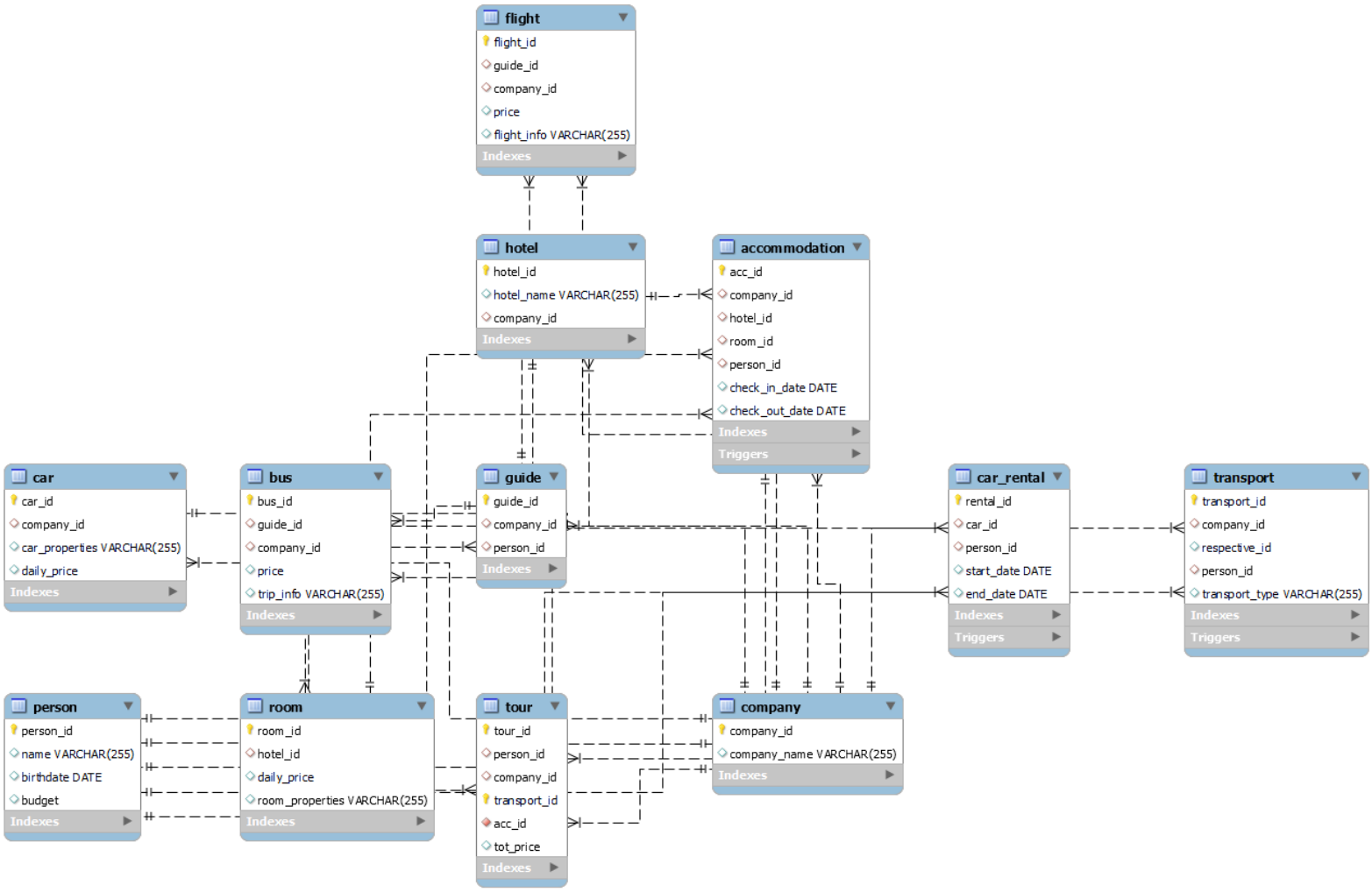
Company

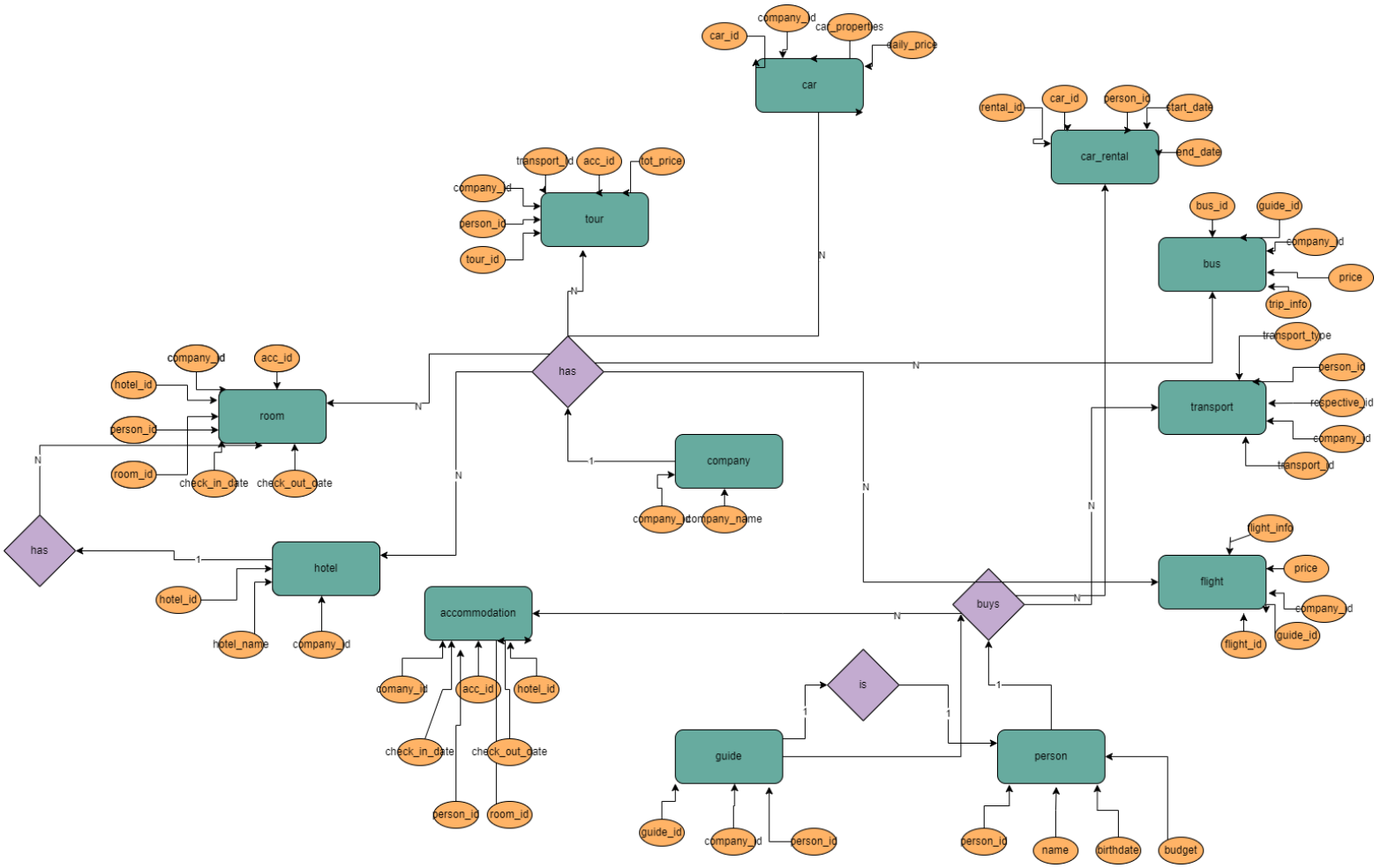
Company can do :

- Insert / Delete / Update car and car rental
- Insert / Delete / Update person
- Insert / Delete / Update accommodation, hotel and room
- Insert / Delete / Update flight, bus trip and its transportations
- Insert / Delete / Update tours
- Insert / Delete / Update guides

And to manage companies, I've added Insert / Delete / Update company.

ER DIAGRAM


flight
flight_id
guide_id
company_id
price
flight_info VARCHAR(255)
Indexes
hotel
hotel_id
hotel_name VARCHAR(255)
company_id
Indexes
accommodation
acc_id
company_id
hotel_id
room_id
person_id
check_in_date DATE
check_out_date DATE
Indexes
Triggers
car
car_id
company_id
car_properties VARCHAR(255)
daily_price
Indexes
bus
bus_id
guide_id
company_id
price
trip_info VARCHAR(255)
Indexes
guide
guide_id
company_id
person_id
Indexes
car_rental
rental_id
car_id
person_id
start_date DATE
end_date DATE
Indexes
Triggers
transport
transport_id
company_id
respective_id
person_id
transport_type VARCHAR(255)
Indexes
Triggers
person
person_id
name VARCHAR(255)
birthdate DATE
budget
Indexes
room
room_id
hotel_id
daily_price
room_properties VARCHAR(255)
Indexes
tour
tour_id
person_id
company_id
transport_id
acc_id
tot_price
Indexes
company
company_id
company_name VARCHAR(255)
Indexes


car
company_id
car_id
car_properties
daily_price
car_rental
rental_id
car_id
person_id
start_date
end_date
tour
transport_id
acc_id
tot_price
company_id
person_id
tour_id
bus
bus_id
guide_id
company_id
price
trip_info
transport
transport_type
person_id
respective_id
company_id
transport_id
has
room
company_id
acc_id
hotel_id
person_id
room_id
check_in_date
check_out_date
company
company_id
company_name
has
hotel
hotel_id
hotel_name
company_id
accommodation
company_id
acc_id
hotel_id
check_in_date
check_out_date
person_id
room_id
buys
flight
flight_info
price
company_id
guide_id
flight_id
is
guide
guide_id
company_id
person_id
person
person_id
name
birthdate
budget

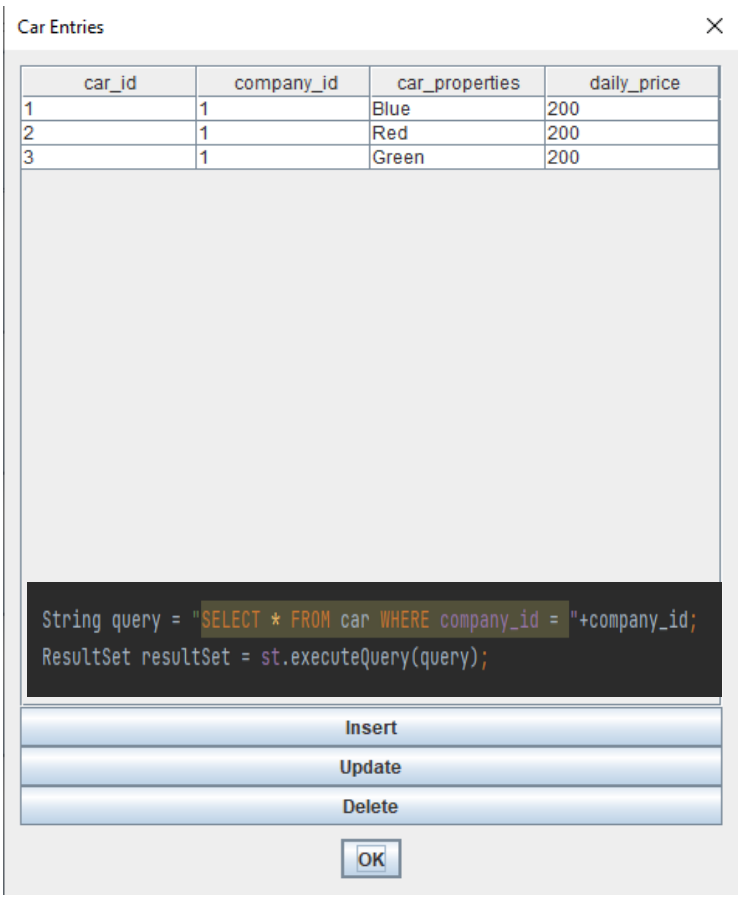
TABLES

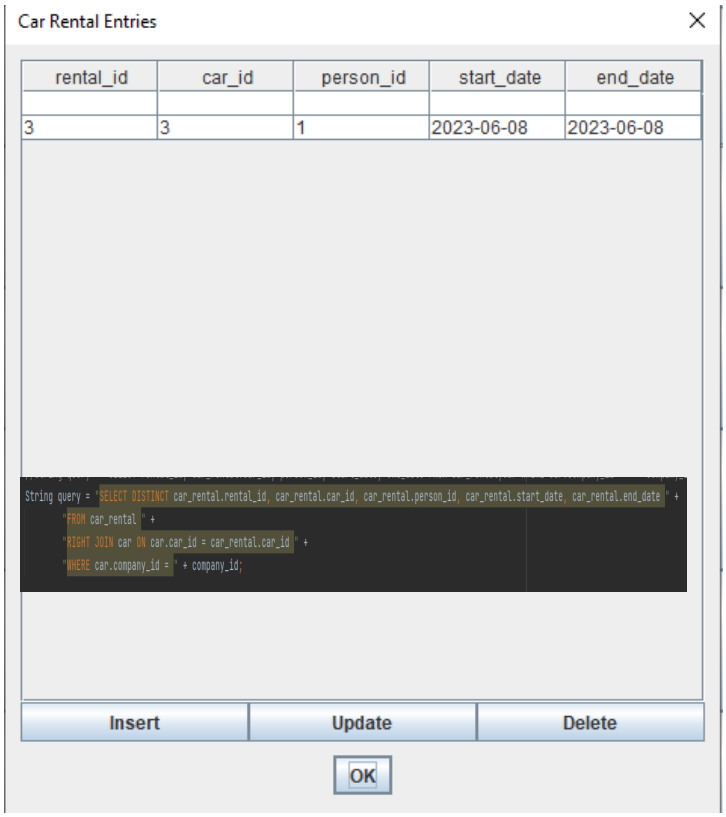
```sql
CREATE TABLE `accommodation` (
  `acc_id` int unsigned NOT NULL AUTO_INCREMENT,
  `company_id` int DEFAULT NULL,
  `hotel_id` int DEFAULT NULL,
  `room_id` int DEFAULT NULL,
  `person_id` int DEFAULT NULL,
  `check_in_date` date DEFAULT NULL,
  `check_out_date` date DEFAULT NULL,
  PRIMARY KEY (`acc_id`),
  UNIQUE KEY `acc_id_UNIQUE` (`acc_id`),
  KEY `company_id` (`company_id`),
  KEY `hotel_id` (`hotel_id`),
  KEY `room_id` (`room_id`),
  KEY `person_id` (`person_id`),
  CONSTRAINT `accommodation_ibfk_1` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`),
  CONSTRAINT `accommodation_ibfk_2` FOREIGN KEY (`hotel_id`) REFERENCES `hotel` (`hotel_id`),
  CONSTRAINT `accommodation_ibfk_3` FOREIGN KEY (`room_id`) REFERENCES `room` (`room_id`),
  CONSTRAINT `accommodation_ibfk_4` FOREIGN KEY (`person_id`) REFERENCES `person` (`person_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `bus` (
  `bus_id` int NOT NULL,
  `guide_id` int DEFAULT NULL,
  `company_id` int DEFAULT NULL,
  `price` int DEFAULT NULL,
  `trip_info` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`bus_id`),
  KEY `guide_id` (`guide_id`),
  KEY `company_id` (`company_id`),
  CONSTRAINT `bus_ibfk_1` FOREIGN KEY (`guide_id`) REFERENCES `guide` (`guide_id`),
  CONSTRAINT `bus_ibfk_2` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `car` (
  `car_id` int NOT NULL,
  `company_id` int DEFAULT NULL,
  `car_properties` varchar(255) DEFAULT NULL,
  `daily_price` int DEFAULT NULL,
  PRIMARY KEY (`car_id`),
  KEY `company_id` (`company_id`),
  CONSTRAINT `car_ibfk_1` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `car_rental` (
  `rental_id` int NOT NULL AUTO_INCREMENT,
  `car_id` int DEFAULT NULL,
  `person_id` int DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `end_date` date DEFAULT NULL,
  PRIMARY KEY (`rental_id`),
  KEY `car_id` (`car_id`),
  KEY `person_id` (`person_id`),
  CONSTRAINT `car_rental_ibfk_1` FOREIGN KEY (`car_id`) REFERENCES `car` (`car_id`),
  CONSTRAINT `car_rental_ibfk_2` FOREIGN KEY (`person_id`) REFERENCES `person` (`person_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `company` (
  `company_id` int NOT NULL,
  `company_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

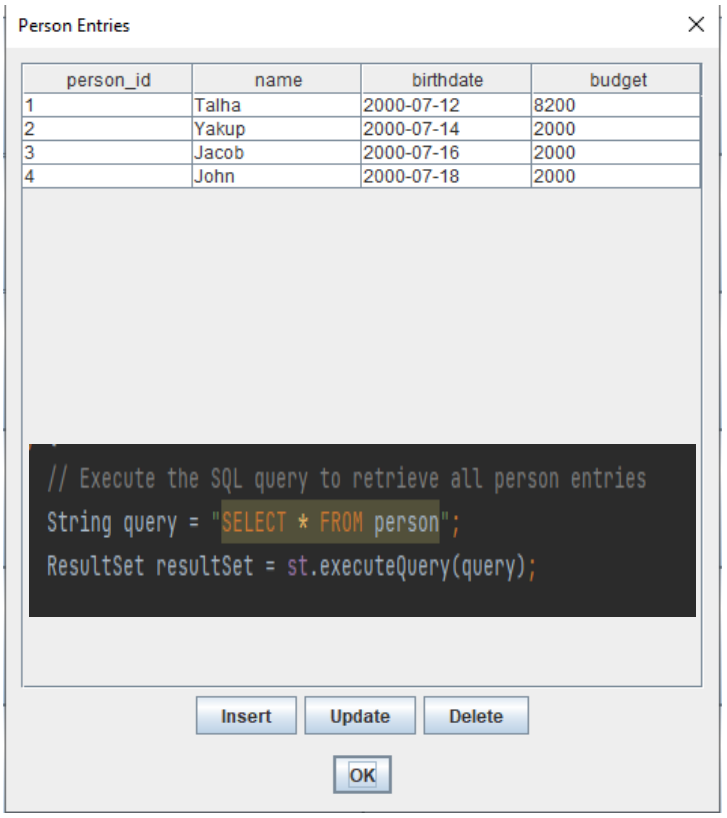
```sql
CREATE TABLE `flight` (
  `flight_id` int NOT NULL,
  `guide_id` int DEFAULT NULL,
  `company_id` int DEFAULT NULL,
  `price` int DEFAULT NULL,
  `flight_info` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`flight_id`),
  KEY `guide_id` (`guide_id`),
  KEY `company_id` (`company_id`),
  CONSTRAINT `flight_ibfk_1` FOREIGN KEY (`guide_id`) REFERENCES `guide` (`guide_id`),
  CONSTRAINT `flight_ibfk_2` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `guide` (
  `guide_id` int NOT NULL,
  `company_id` int DEFAULT NULL,
  `person_id` int DEFAULT NULL,
  PRIMARY KEY (`guide_id`),
  KEY `company_id` (`company_id`),
  KEY `person_id` (`person_id`),
  CONSTRAINT `guide_ibfk_1` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`),
  CONSTRAINT `guide_ibfk_2` FOREIGN KEY (`person_id`) REFERENCES `person` (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `hotel` (
  `hotel_id` int NOT NULL,
  `hotel_name` varchar(255) DEFAULT NULL,
  `company_id` int DEFAULT NULL,
  PRIMARY KEY (`hotel_id`),
  KEY `company_id` (`company_id`),
  CONSTRAINT `hotel_ibfk_1` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

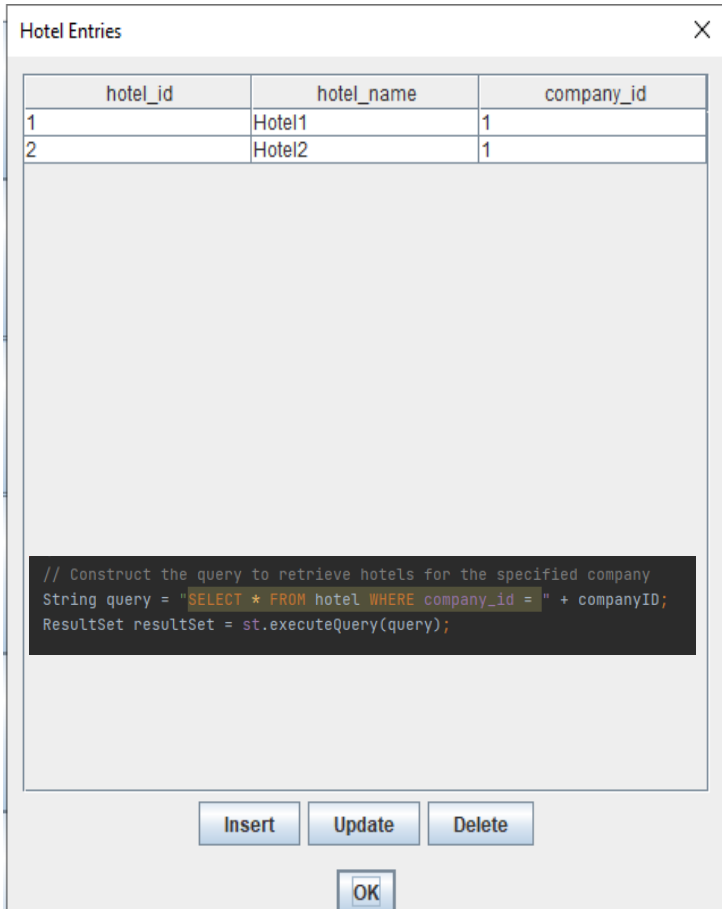
```sql
CREATE TABLE `person` (
  `person_id` int NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `birthdate` date DEFAULT NULL,
  `budget` int DEFAULT NULL,
  PRIMARY KEY (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `room` (
  `room_id` int NOT NULL,
  `hotel_id` int DEFAULT NULL,
  `daily_price` int DEFAULT NULL,
  `room_properties` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`room_id`),
  KEY `hotel_id` (`hotel_id`),
  CONSTRAINT `room_ibfk_1` FOREIGN KEY (`hotel_id`) REFERENCES `hotel` (`hotel_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `tour` (
  `tour_id` int NOT NULL,
  `person_id` int DEFAULT NULL,
  `company_id` int DEFAULT NULL,
  `transport_id` int unsigned NOT NULL,
  `acc_id` int unsigned NOT NULL,
  `tot_price` int DEFAULT NULL,
  PRIMARY KEY (`tour_id`,`transport_id`),
  KEY `person_id` (`person_id`),
  KEY `company_id` (`company_id`),
  KEY `ibfk4` (`transport_id`),
  KEY `ibfk5` (`acc_id`),
  CONSTRAINT `ibfk5` FOREIGN KEY (`acc_id`) REFERENCES `accommodation` (`acc_id`),
  CONSTRAINT `tour_ibfk_1` FOREIGN KEY (`person_id`) REFERENCES `person` (`person_id`),
  CONSTRAINT `tour_ibfk_2` FOREIGN KEY (`company_id`) REFERENCES `company` (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

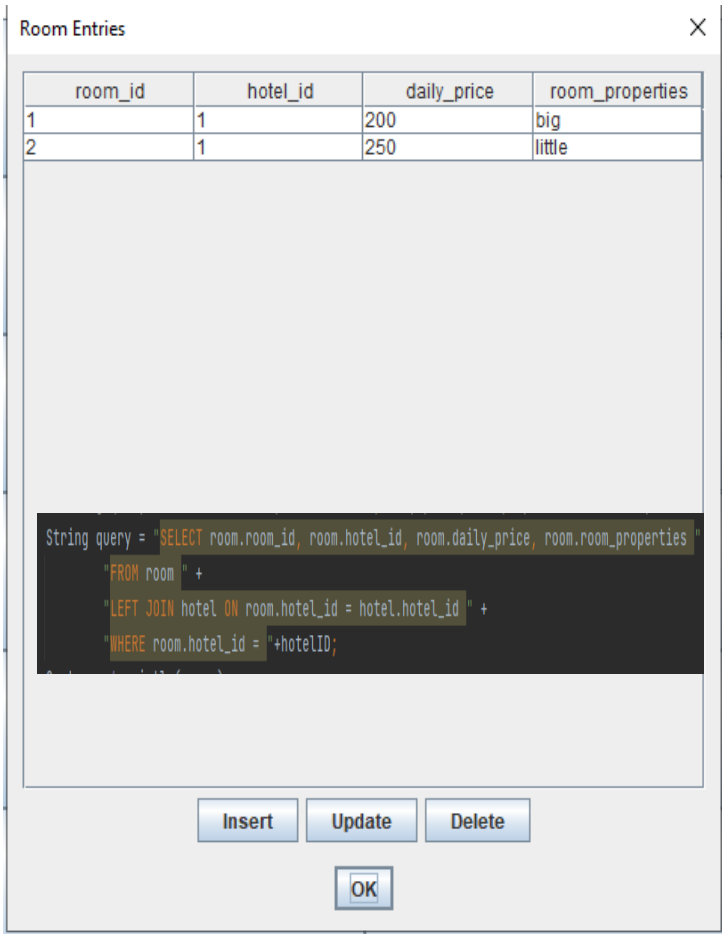
SELECT EXAMPLE

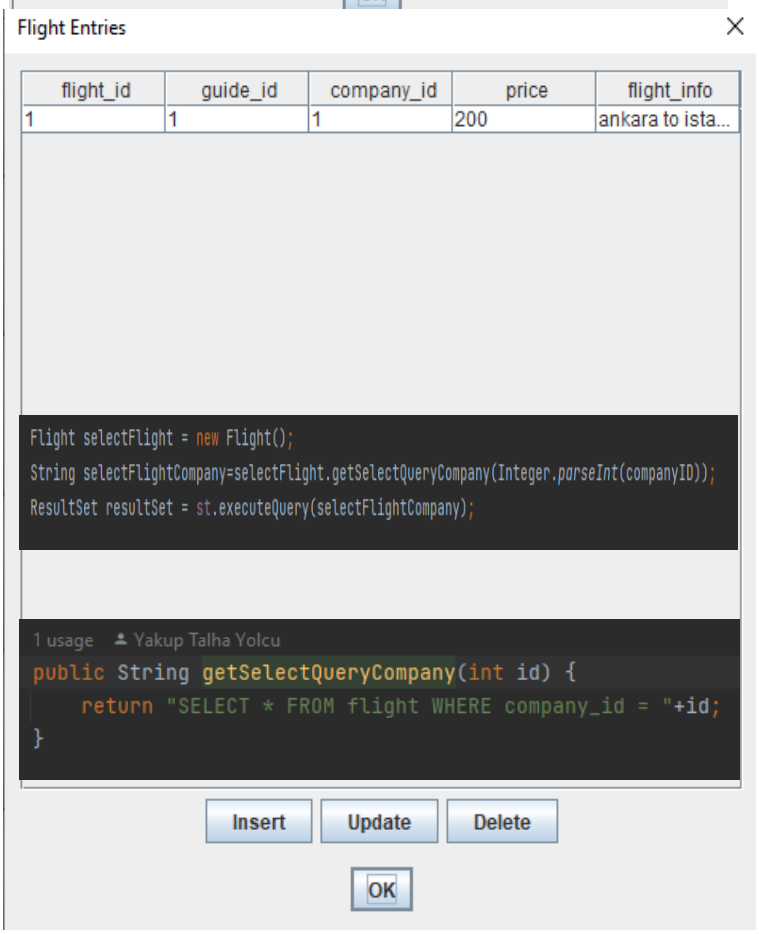




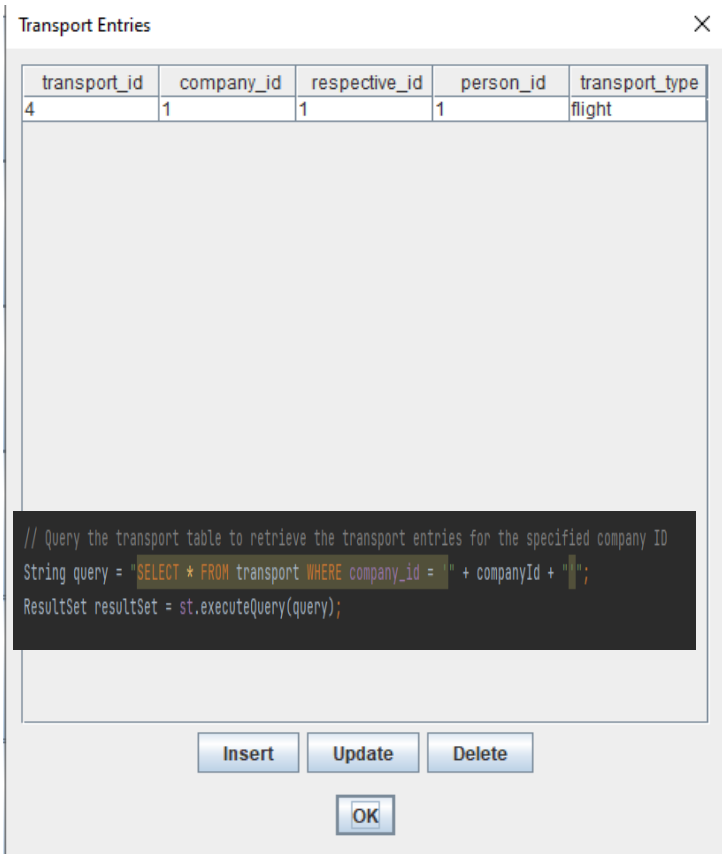


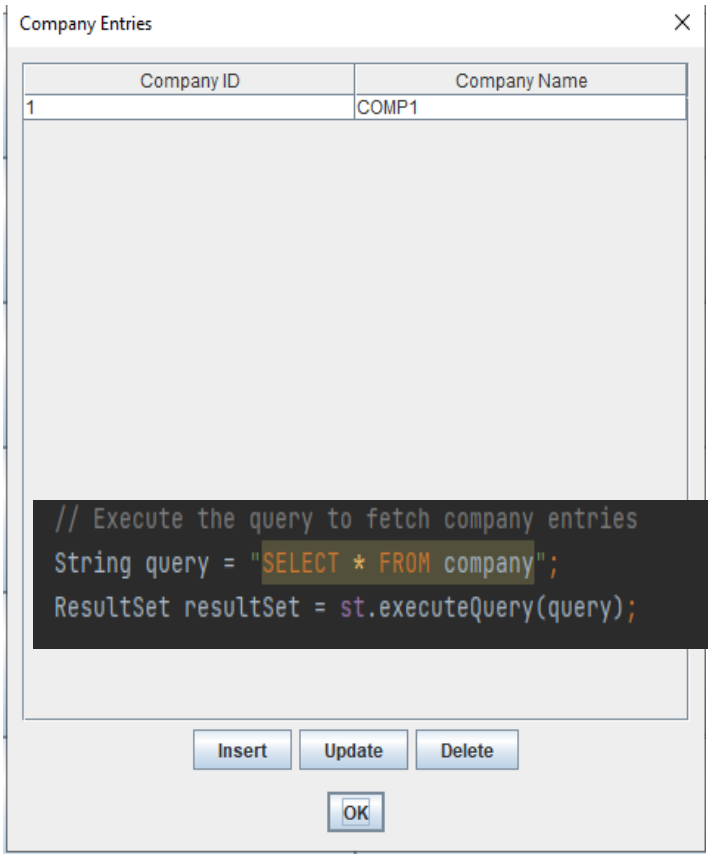

Hotel Entries
hotel_id
hotel_name
company_id
1
Hotel1
1
2
Hotel2
1
// Construct the query to retrieve hotels for the specified company
String query = "SELECT * FROM hotel WHERE company_id = " + companyID;
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK


Room Entries
room_id
hotel_id
daily_price
room_properties
1
1
200
big
2
1
250
little
String query = "SELECT room.room_id, room.hotel_id, room.daily_price, room.room_properties "
"FROM room " +
"LEFT JOIN hotel ON room.hotel_id = hotel.hotel_id " +
"WHERE room.hotel_id = "+hotelID;
Insert
Update
Delete
OK


Flight Entries
flight_id
guide_id
company_id
price
flight_info
1
1
1
200
ankara to ista...
Flight selectFlight = new Flight();
String selectFlightCompany=selectFlight.getSelectQueryCompany(Integer.parseInt(companyID));
ResultSet resultSet = st.executeQuery(selectFlightCompany);
1 usage
Yakup Talha Yolcu
public String getSelectQueryCompany(int id) {
return "SELECT * FROM flight WHERE company_id = "+id;
}
Insert
Update
Delete
OK


Bus Entries
bus_id
guide_id
company_id
price
trip_info
1
1
1
200
istanbul to an...
// Construct the query to retrieve buses for the specified company
String query = "SELECT * FROM bus WHERE company_id = " + companyID;
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK


Transport Entries
transport_id
company_id
respective_id
person_id
transport_type
4
1
1
1
flight
// Query the transport table to retrieve the transport entries for the specified company ID
String query = "SELECT * FROM transport WHERE company_id = '" + companyId + "'";
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK

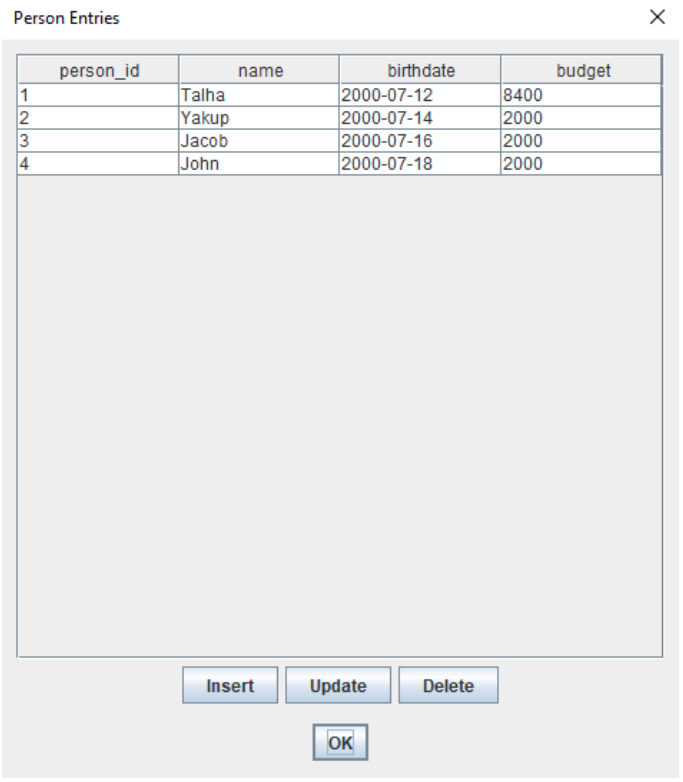

Company Entries
Company ID
Company Name
1
COMP1
// Execute the query to fetch company entries
String query = "SELECT * FROM company";
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK

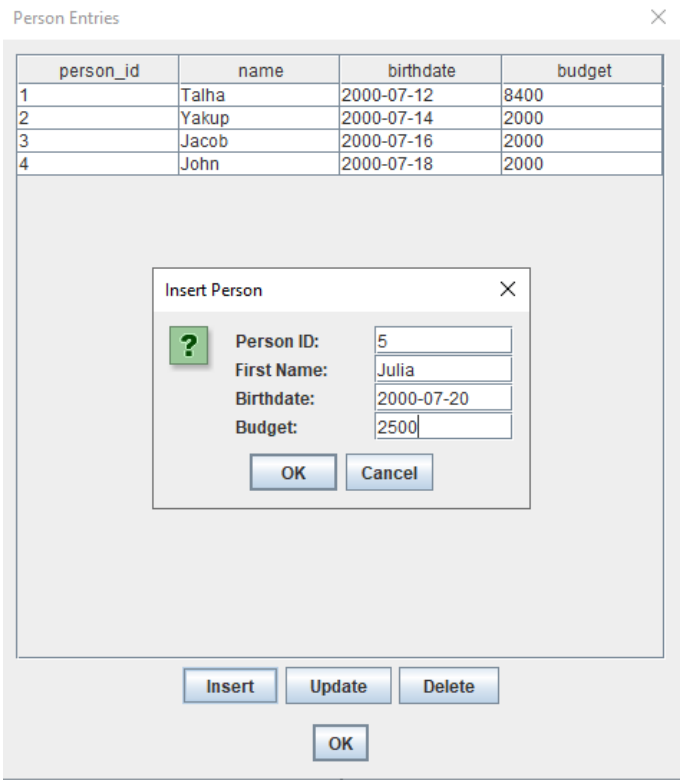

View Tours
Tour ID
Person ID
Company ID
Accommod...
Transport ID
Total Price
1
1
1
4
4
500
// Execute the query to fetch tour entries for the specified company ID
String query = "SELECT * FROM tour WHERE company_id = " + companyID;
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK


View Guides
guide_id
company_id
person_id
1
1
2
// Execute the query to retrieve the guides for the specified company ID
String query = "SELECT * FROM guide WHERE company_id = '" + companyId + "'";
ResultSet resultSet = st.executeQuery(query);
Insert
Update
Delete
OK

INSERT EXAMPLE - 1

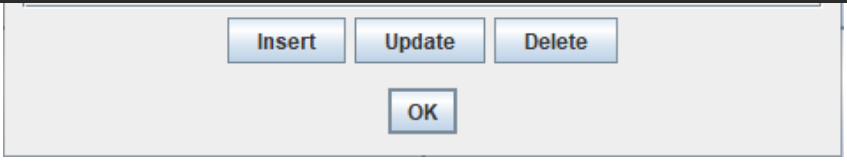
Company inserts a person





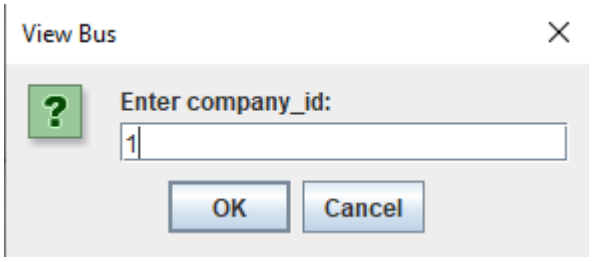

```
Person person = new Person(Integer.parseInt(personId),firstName,date,Integer.parseInt(budget));
String query = person.getInsertQuery(person);
```

```
@Override
public String getInsertQuery(Person object) {
    return "INSERT INTO person (person_id, name, birthdate, budget) VALUES (" +
            "" + object.getPerson_id() + "," +
            "'" + object.getName() + "'," +
            "'" + object.getBirthdate() + "'," +
            "" + object.getBudget() +
            ")";
}
```

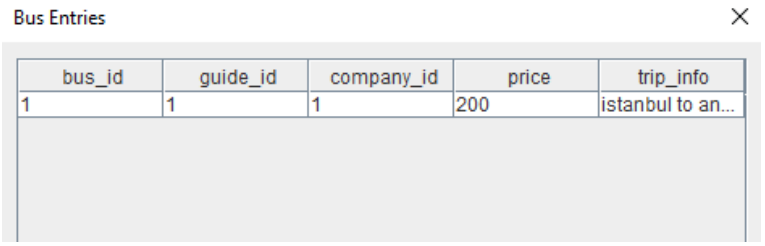


INSERT EXAMPLE - 2

Company inserts bus
Enter company id

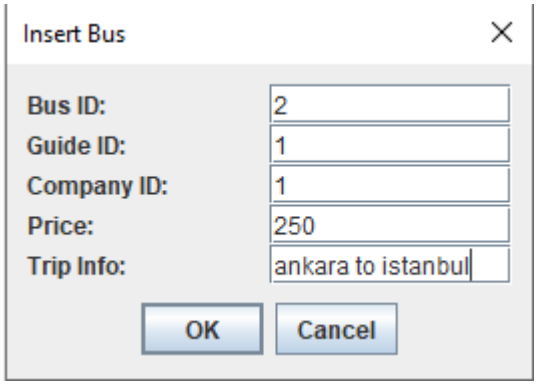


Before insert

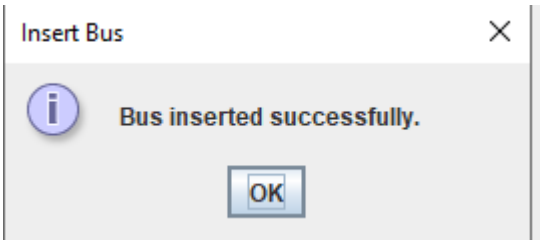


Enter bus table info

While inserting :

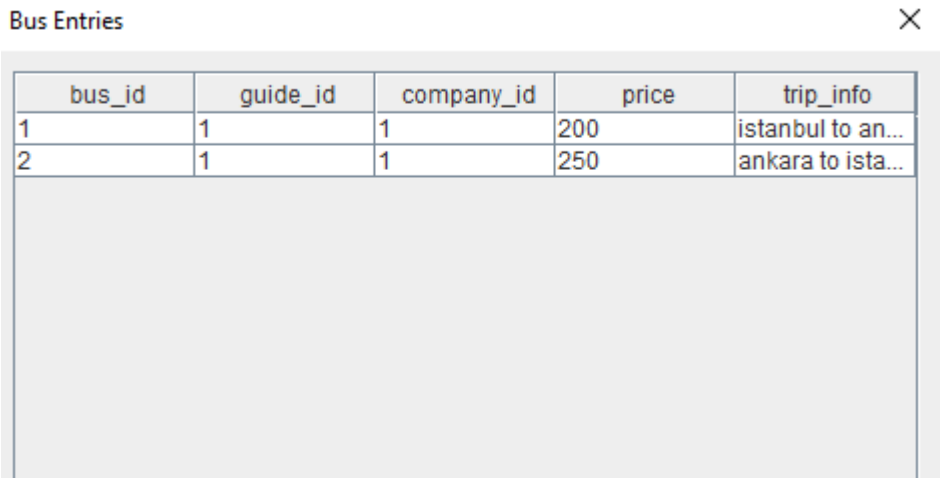



```
Bus bus = new Bus(
  Integer.parseInt(busId),Integer.parseInt(guideId),
  Integer.parseInt(companyId),
  Integer.parseInt(price),tripInfo
);

String insertQuery = bus.getInsertQuery(bus);
```




```
@Override
public String getInsertQuery(Bus object) {
    return "INSERT INTO bus (bus_id, guide_id, company_id, price, trip_info) VALUES (" +
            "" + object.getBus_id() + "," +
            "" + object.getGuide_id() + "," +
            "" + object.getCompany_id() + "," +
            "" + object.getPrice() + "," +
            "'" + object.getTrip_info() +
            "')";
}
```

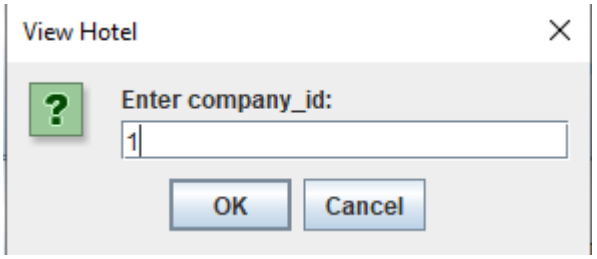
After insert :



INSERT EXAMPLE - 3

Company inserts hotel
Enter company_id

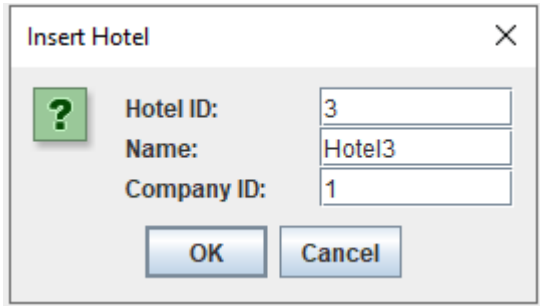
Before insert



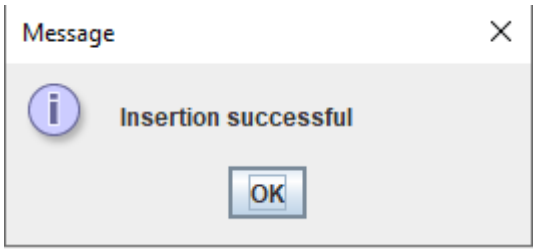
Hotel Entries

hotel_id	hotel_name	company_id
1	Hotel1	1
2	Hotel2	1

While inserting :




```
Hotel hotel = new Hotel(Integer.parseInt(hotelId),name,Integer.parseInt(companyId));
String query = hotel.getInsertQuery(hotel);
```




```
@Override
public String getInsertQuery(Hotel object) {
    return "INSERT INTO hotel (hotel_id, hotel_name, company_id) VALUES (" +
            "" + object.getHotel_id() + "," +
            "'" + object.getHotel_name() + "'," +
            "" + object.getCompany_id() +
            ")";
}
```

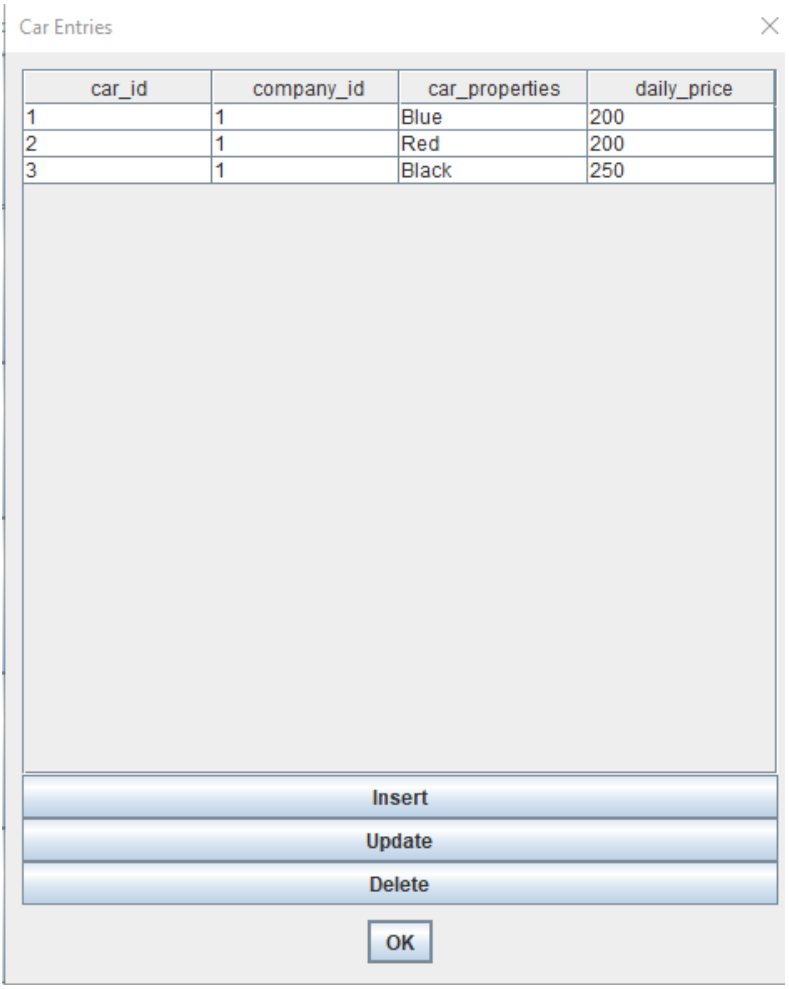
After insert :

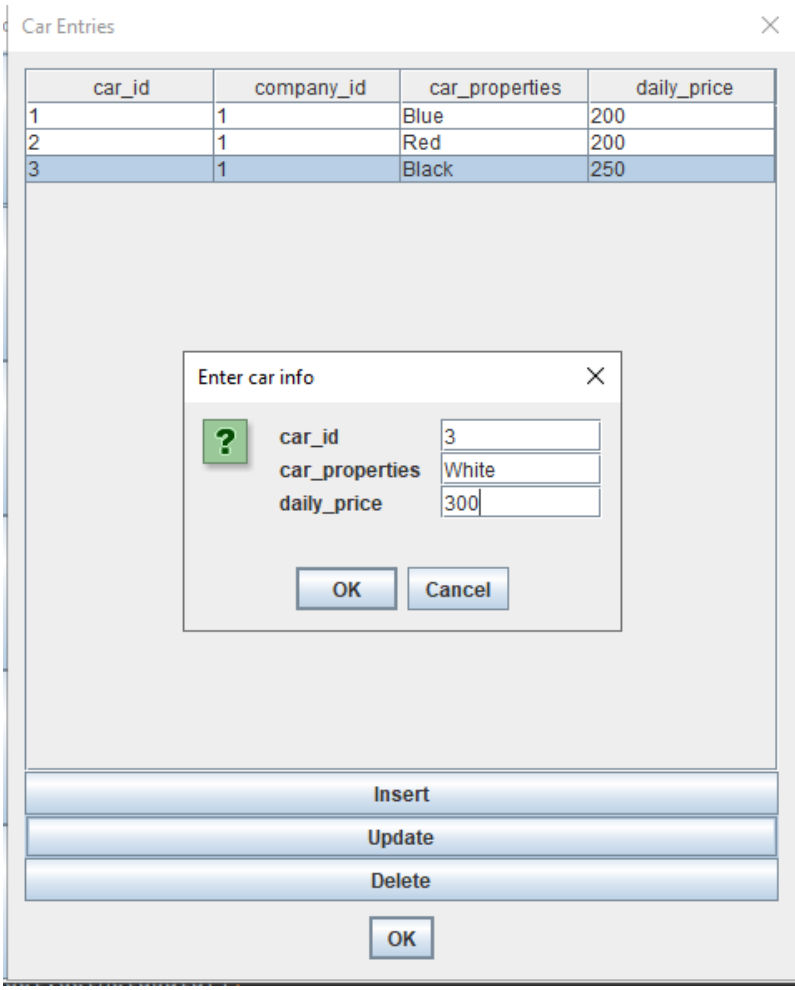
Hotel Entries

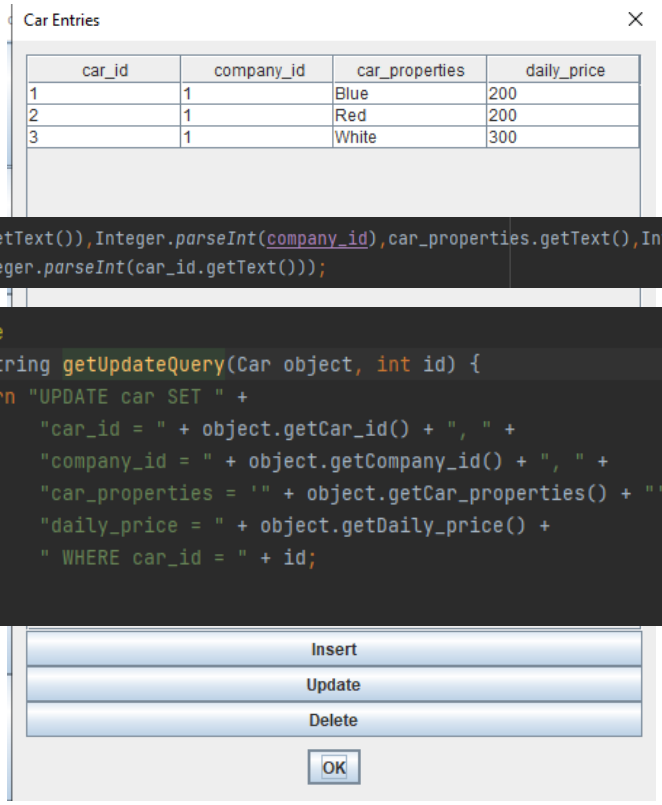
hotel_id	hotel_name	company_id
1	Hotel1	1
2	Hotel2	1
3	Hotel3	1

UPDATE EXAMPLE - 1

Company updates a car





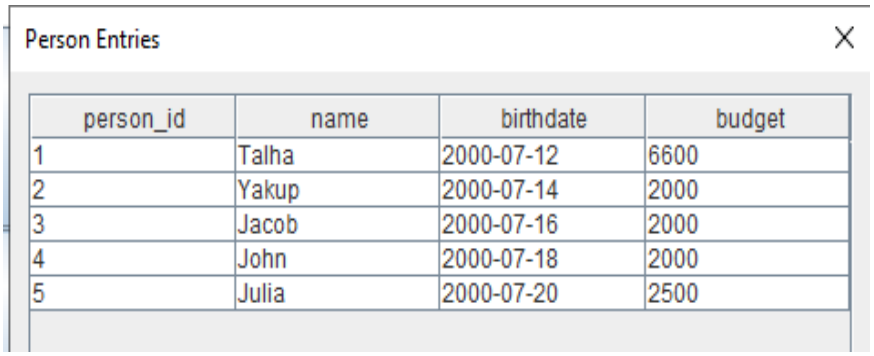


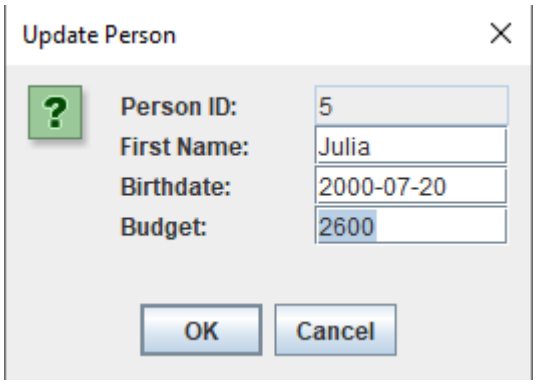

```
Car car=new Car(Integer.parseInt(car_id.getText()),Integer.parseInt(company_id),car_properties.getText(),Integer.parseInt(daily_price.getText()));
String query = car.getUpdateQuery(car,Integer.parseInt(car_id.getText()));
```

```
@Override
public String getUpdateQuery(Car object, int id) {
    return "UPDATE car SET " +
            "car_id = " + object.getCar_id() + ", " +
            "company_id = " + object.getCompany_id() + ", " +
            "car_properties = '" + object.getCar_properties() + "', " +
            "daily_price = " + object.getDaily_price() +
            " WHERE car_id = " + id;
}
```

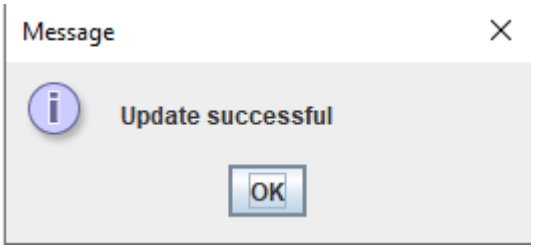
UPDATE EXAMPLE - 2

Updating a person
Before update



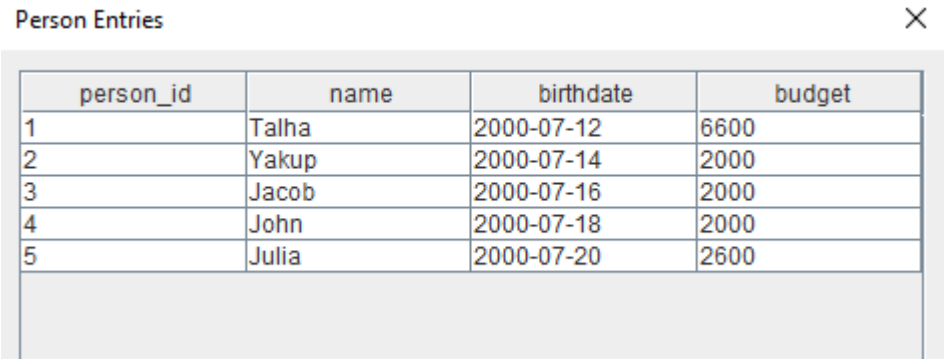



```
Person person = new Person(Integer.parseInt(personId),updatedFirstName,Date.valueOf(updatedAge),Integer.parseInt(updatedBudget));
String query = person.getUpdateQuery(person,Integer.parseInt(personId));
System.out.println(query);
```



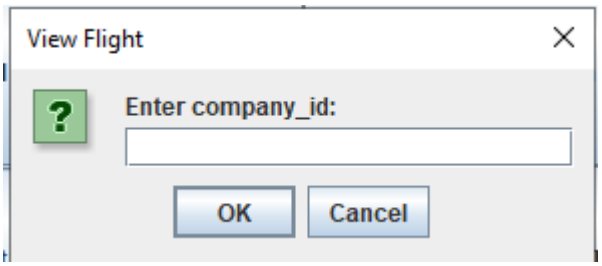

```
@Override
public String getUpdateQuery(Person object, int id) {
    return "UPDATE person SET " +
            "person_id = " + object.getPerson_id() + ", " +
            "name = '" + object.getName() + "', " +
            "birthdate = '" + object.getBirthdate() + "', " +
            "budget = " + object.getBudget() +
            " WHERE person_id = " + id;
}
```

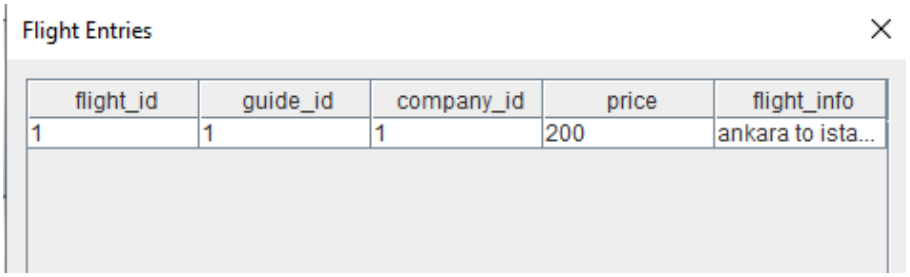
After update



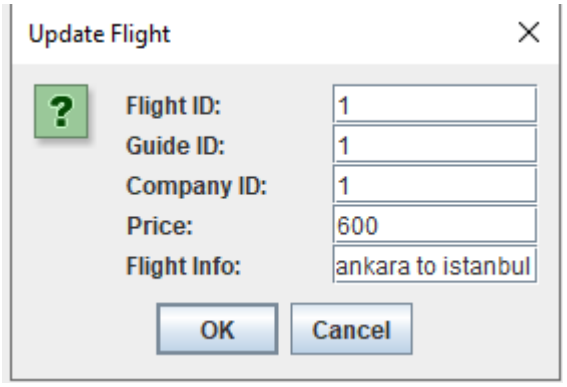
UPDATE EXAMPLE - 3

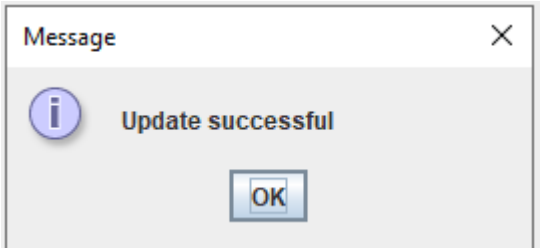
Company updates flight
Enter company_id



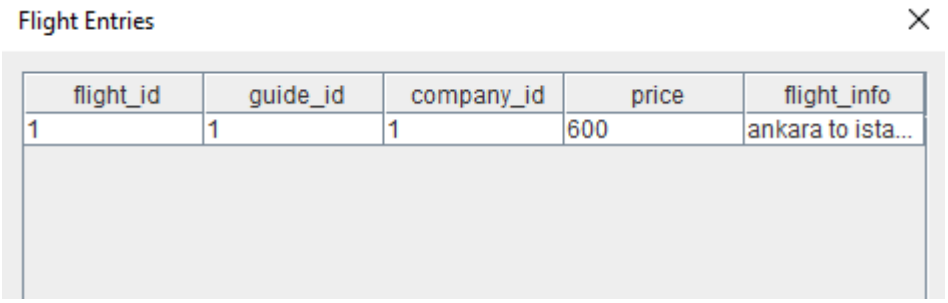


While updating :





After update :

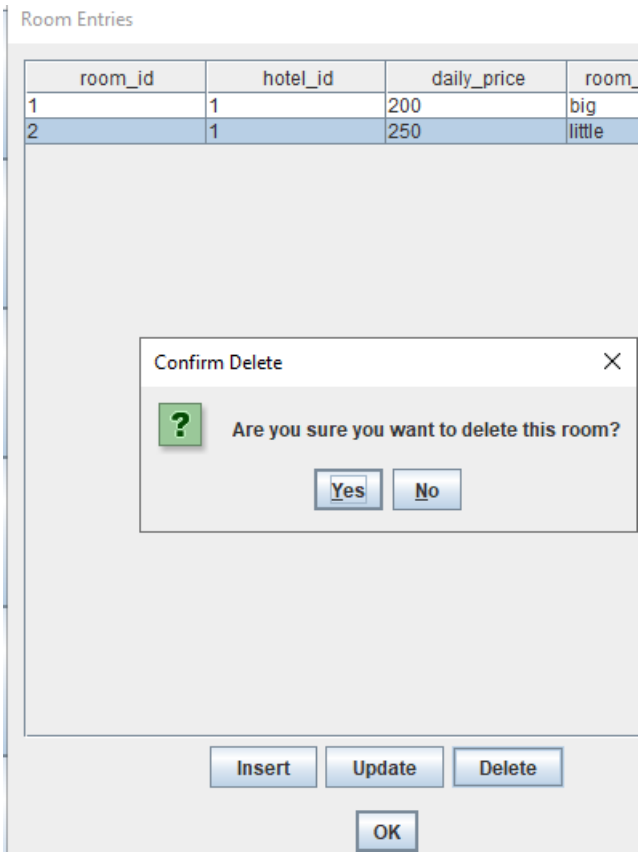


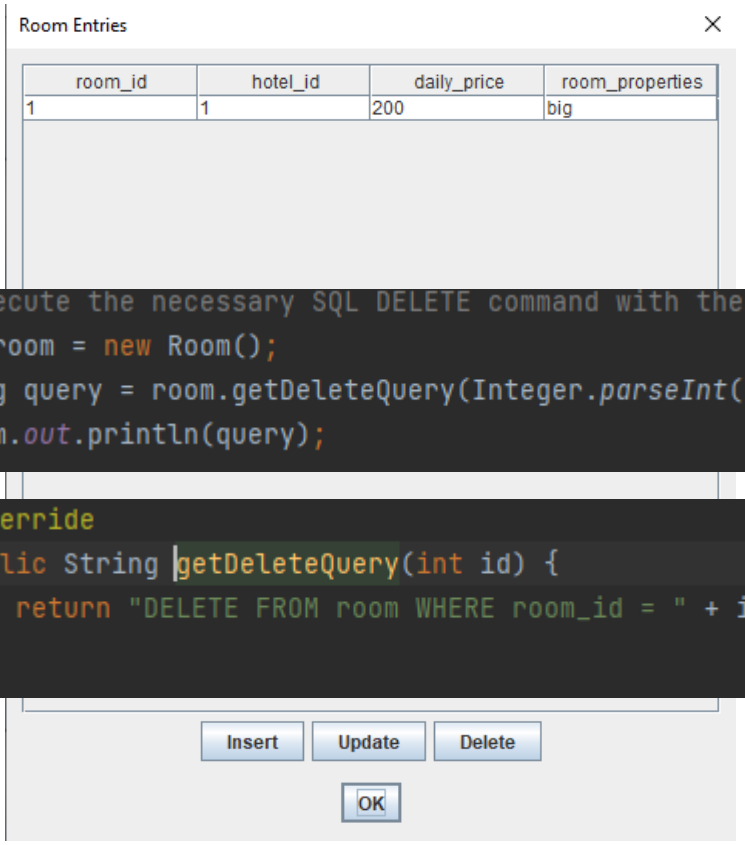

```
Flight flight = new Flight(
  Integer.parseInt(updatedFlightId),Integer.parseInt(updatedGuideId),
  Integer.parseInt(updatedCompanyId),Integer.parseInt(updatedPrice),
  updatedFlightInfo
);
String query = flight.getUpdateQuery(flight,Integer.parseInt(updatedFlightId));
System.out.println(query);
```

```
@Override
public String getUpdateQuery(Flight object, int id) {
    return "UPDATE flight SET " +
            "flight_id = " + object.getFlight_id() + ", " +
            "guide_id = " + object.getGuide_id() + ", " +
            "company_id = " + object.getCompany_id() + ", " +
            "price = " + object.getPrice() + ", " +
            "flight_info = '" + object.getFlight_info() +
            "' WHERE flight_id = " + id;
}
```

DELETE EXAMPLE - 1

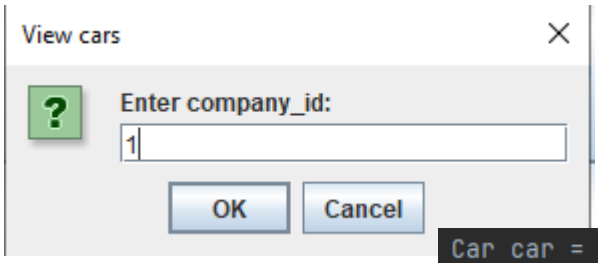
Company deletes a hotel room





DELETE EXAMPLE - 2

Company deletes a car
Enter company_id



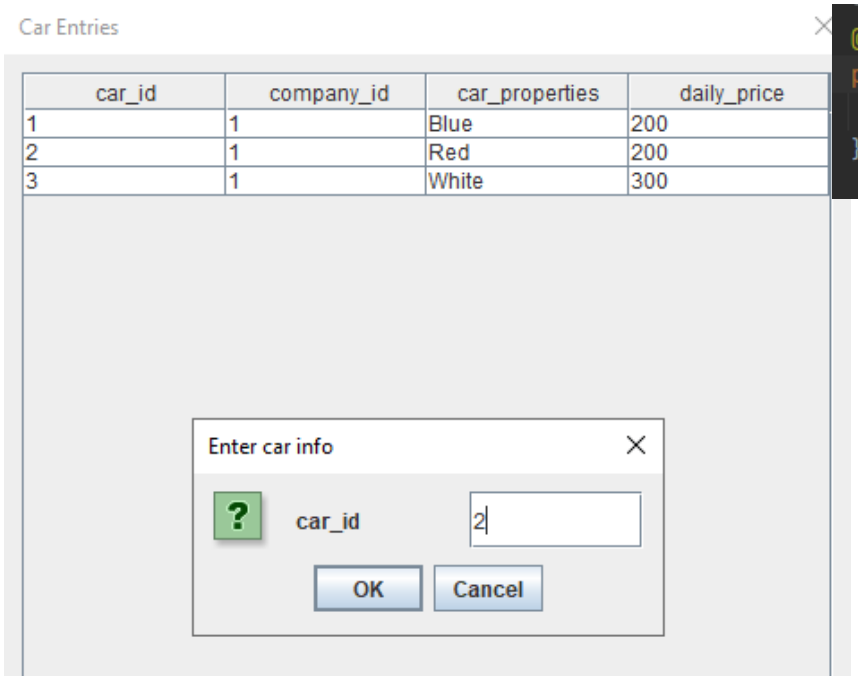

```
Car car = new Car();
String query = car.getDeleteQuery(Integer.parseInt(car_id.getText()));
```

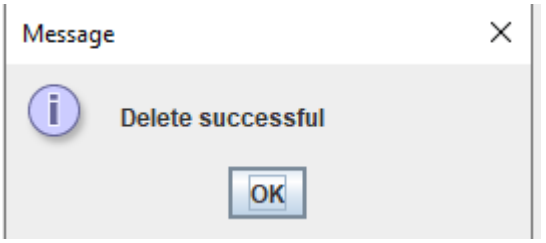
Before delete

Car Entries

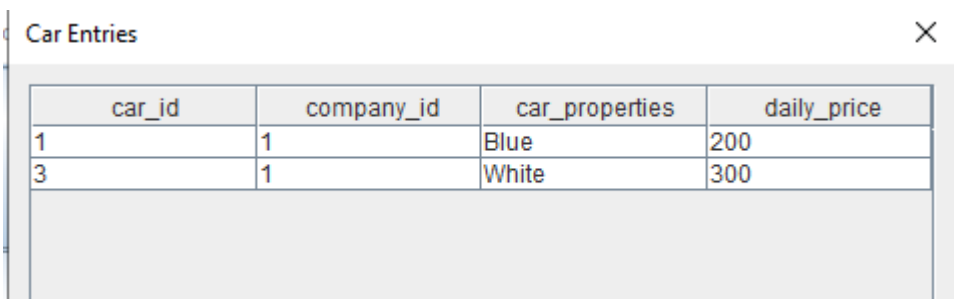
car_id	company_id	car_properties	daily_price
1	1	Blue	200
2	1	Red	200
3	1	White	300

```
@Override
public String getDeleteQuery(int id) {
    return "DELETE FROM car WHERE car_id = " + id;
}
```



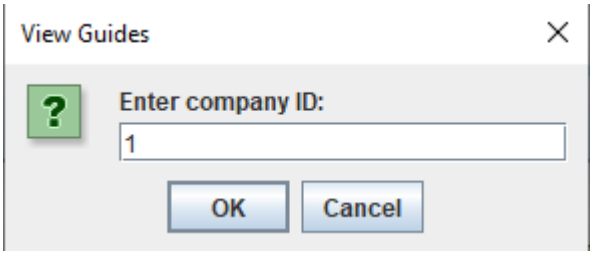


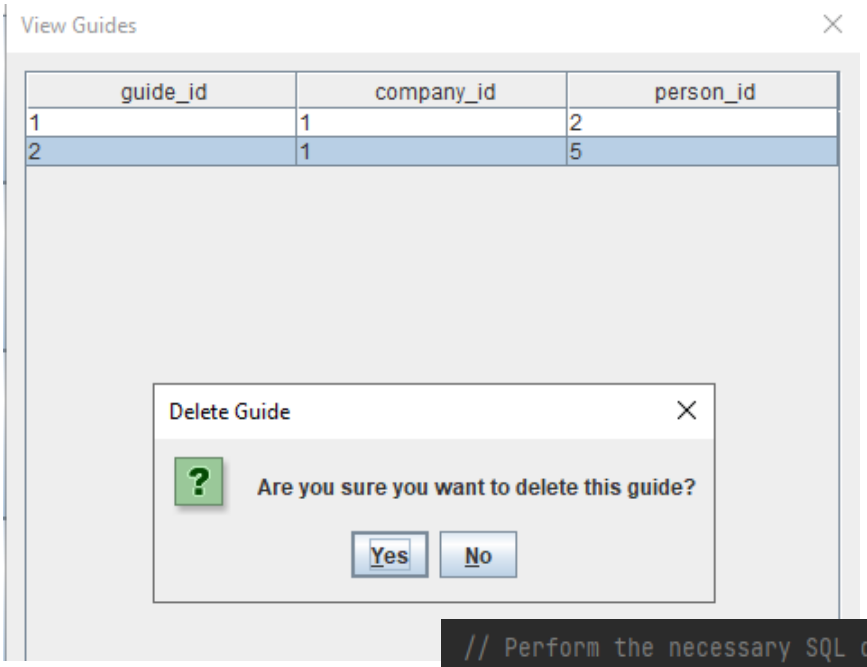
After delete



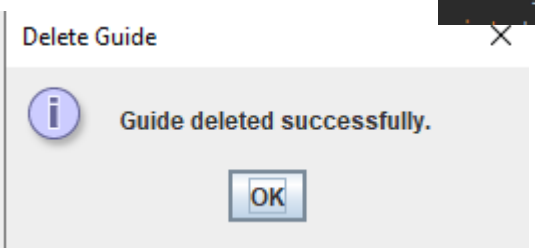
DELETE EXAMPLE - 3

Deleting a guide
Enter company_id






```
// Perform the necessary SQL delete operation
String deleteQuery = new Guide().getDeleteQuery(Integer.parseInt(guideId));
```




```
@Override
public String getDeleteQuery(int id) {
    return "DELETE FROM guide WHERE guide_id = " + id;
}
```

After delete

FUNCTIONAL DEPENDENCIES

Table: accommodation
acc_id -> company_id, hotel_id, room_id, person_id, check_in_date, check_out_date

Table: bus
bus_id -> guide_id, company_id, price, trip_info

Table: car
car_id -> company_id, car_properties, daily_price

Table: car_rental
rental_id -> car_id, person_id, start_date, end_date

Table: company
company_id -> company_name

Table: flight
flight_id -> guide_id, company_id, price, flight_info

Table: guide
guide_id -> company_id, person_id

Table: hotel
hotel_id -> hotel_name, company_id

Table: person
person_id -> name, birthdate, budget

Table: room
room_id -> hotel_id, daily_price, room_properties

Table: transport
transport_id -> company_id, respective_id, person_id, transport_type

CHECKING NORMALIZATION

Table: accommodation
Functional dependencies: acc_id -> company_id, hotel_id, room_id, person_id, check_in_date, check_out_date

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and acc_id is a primary key.

Table: bus
Functional dependencies: bus_id -> guide_id, company_id, price, trip_info

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and bus_id is a primary key.

Table: car
Functional dependencies: car_id -> company_id, car_properties, daily_price

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and car_id is a primary key.

Table: car_rental
Functional dependencies: rental_id -> car_id, person_id, start_date, end_date

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and rental_id is a primary key.

Table: company
Functional dependencies: company_id -> company_name

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and company_id is a primary key.

Table: flight
Functional dependencies: flight_id -> guide_id, company_id, price, flight_info

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and flight_id is a primary key.

Table: guide
Functional dependencies: guide_id -> company_id, person_id

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and guide_id is a primary key.

Table: hotel
Functional dependencies: hotel_id -> hotel_name, company_id

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and hotel_id is a primary key.

Table: person
Functional dependencies: person_id -> name, birthdate, budget

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and person_id is a primary key.

Table: room
Functional dependencies: room_id -> hotel_id, daily_price, room_properties

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and room_id is a primary key.

Table: transport
Functional dependencies: transport_id -> company_id, respective_id, person_id, transport_type

This table does not violate BCNF or 3NF. The functional dependency does not have any trivial dependencies, and transport_id is a primary key.

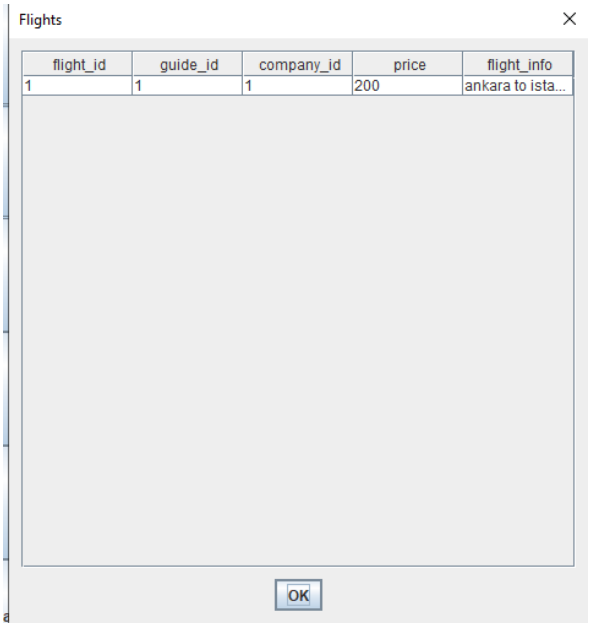
Based on the evaluation, all the tables satisfy BCNF and 3NF. Therefore, no decomposition is required to achieve these normalization forms.

VIEWS

I've created my views like that :

```
try {
    st = TravelBookingConnection.getConnection().createStatement();
    st.execute( sql: "CREATE VIEW select_car AS" +
            " SELECT car_id,company_id,car_properties,daily_price FROM car;");
    st.execute( sql: "CREATE VIEW select_car_rental AS SELECT * FROM car_rental;");
    st.execute( sql: "CREATE VIEW select_accommodation AS SELECT * FROM accommodation;");
    st.execute( sql: "CREATE VIEW select_bus AS SELECT * FROM bus;");
    st.execute( sql: "CREATE VIEW select_flight AS SELECT * FROM flight;");
}
catch (SQLException ex) {
    ex.printStackTrace();
}
```

Interface example for a one view:
Person looks for flights:



And queries for that views :

```java
@Override
public String getSelectAllQuery() {
    return "SELECT * FROM select_car";
}
```

```java
@Override
public String getSelectAllQuery() {
    return "SELECT * FROM select_car_rental";
}
```

```java
@Override
public String getSelectAllQuery() { return "SELECT * FROM select_accommodation"; }
```

```java
@Override
public String getSelectAllQuery() { return "SELECT * FROM select_bus"; }
```

```java
@Override
public String getSelectAllQuery() { return "SELECT * FROM select_flight"; }
```

TRIGGERS

Update person budget when accommodation is bought or canceled

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `accommodation_AFTER_INSERT` AFTER INSERT ON `accommodation` FOR EACH ROW BEGIN
    -- Retrieve the daily_price for the room from the room table
    DECLARE room_price INT;
    SELECT daily_price INTO room_price
    FROM room
    WHERE room_id = NEW.room_id AND hotel_id = NEW.hotel_id;

    -- Update the person's budget
    UPDATE person
    SET budget = budget - (room_price * DATEDIFF(NEW.check_out_date, NEW.check_in_date))
    WHERE person_id = NEW.person_id;
END
```

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `accommodation_AFTER_DELETE` AFTER DELETE ON `accommodation` FOR EACH ROW BEGIN
    -- Retrieve the daily_price for the room from the room table
    DECLARE room_price INT;
    SELECT daily_price INTO room_price
    FROM room
    WHERE room_id = OLD.room_id AND hotel_id = OLD.hotel_id;

    -- Update the person's budget
    UPDATE person
    SET budget = budget + (room_price * DATEDIFF(OLD.check_out_date, OLD.check_in_date))
    WHERE person_id = OLD.person_id;
END
```

Update person budget when car rental is bought or canceled

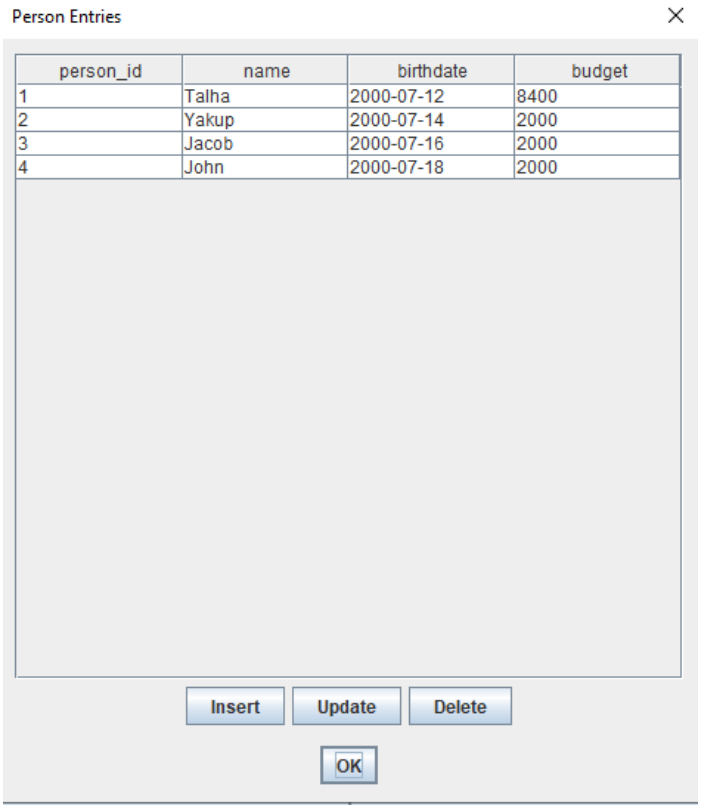
```
CREATE DEFINER=`root`@`localhost` TRIGGER `car_rental_AFTER_INSERT` AFTER INSERT ON `car_rental` FOR EACH ROW BEGIN
    -- Retrieve the daily_price for the car from the Car table
    DECLARE car_price INT;
    SELECT daily_price INTO car_price FROM Car WHERE car_id = NEW.car_id;

    -- Update the person's budget
    UPDATE person
    SET budget = budget - (car_price * DATEDIFF(NEW.end_date, NEW.start_date))
    WHERE person_id = NEW.person_id;
END
```

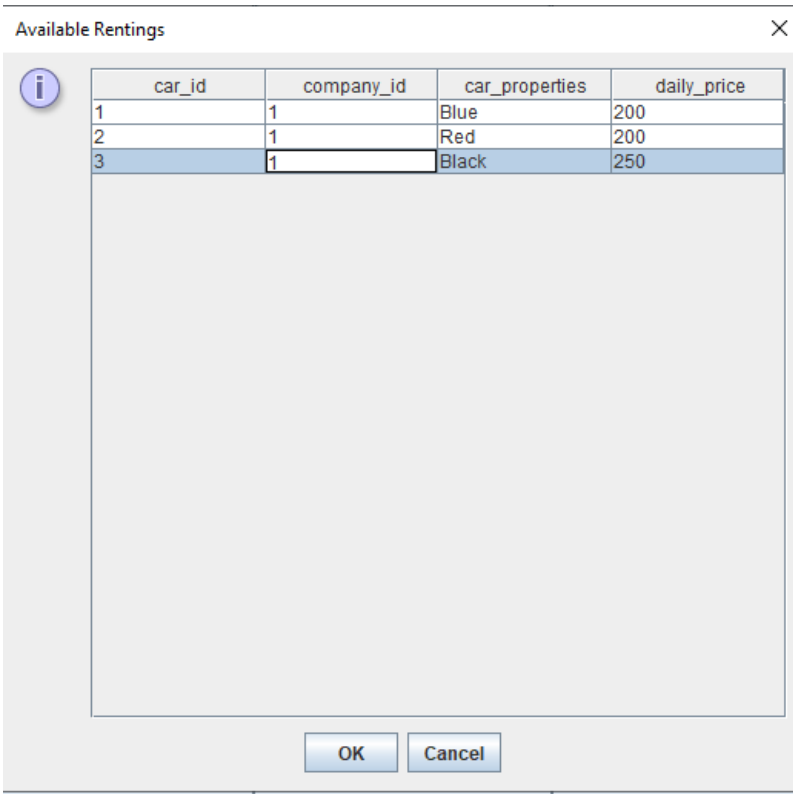
```
CREATE DEFINER=`root`@`localhost` TRIGGER `car_rental_AFTER_DELETE` AFTER DELETE ON `car_rental` FOR EACH ROW BEGIN
    -- Retrieve the car's daily_price from the Car table
    DECLARE car_price INT;
    SELECT daily_price INTO car_price FROM Car WHERE car_id = OLD.car_id;

    -- Update the person's budget
    UPDATE person
    SET budget = budget + (car_price * DATEDIFF(OLD.end_date, OLD.start_date))
    WHERE person_id = OLD.person_id;
END
```

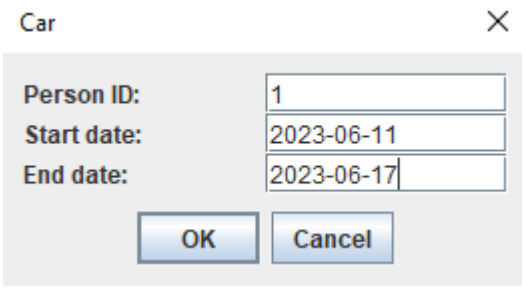
Update person budget when a transport is bought or canceled

```
CREATE DEFINER=`root`@`localhost` TRIGGER `transport_AFTER_INSERT` AFTER INSERT ON `transport` FOR EACH ROW BEGIN
    DECLARE transport_price INT;

    -- Retrieve the price based on the transport_type and respective_id
    IF NEW.transport_type = 'flight' THEN
        SELECT price INTO transport_price
        FROM flight
        WHERE flight_id = NEW.respective_id AND company_id = NEW.company_id;
    ELSEIF NEW.transport_type = 'bus' THEN
        SELECT price INTO transport_price
        FROM bus
        WHERE bus_id = NEW.respective_id AND company_id = NEW.company_id;
    END IF;

    -- Update the person's budget
    UPDATE person
    SET budget = budget - transport_price
    WHERE person_id = NEW.person_id;
END
CREATE DEFINER=`root`@`localhost` TRIGGER `transport_AFTER_DELETE` AFTER DELETE ON `transport` FOR EACH ROW BEGIN
    DECLARE transport_price INT;

    -- Retrieve the price based on the transport_type and respective_id
    IF OLD.transport_type = 'flight' THEN
        SELECT price INTO transport_price
        FROM flight
        WHERE flight_id = OLD.respective_id AND company_id = OLD.company_id;
    ELSEIF OLD.transport_type = 'bus' THEN
        SELECT price INTO transport_price
        FROM bus
        WHERE bus_id = OLD.respective_id AND company_id = OLD.company_id;
    END IF;

    -- Update the person's budget
    UPDATE person
    SET budget = budget + transport_price
    WHERE person_id = OLD.person_id;
END
```

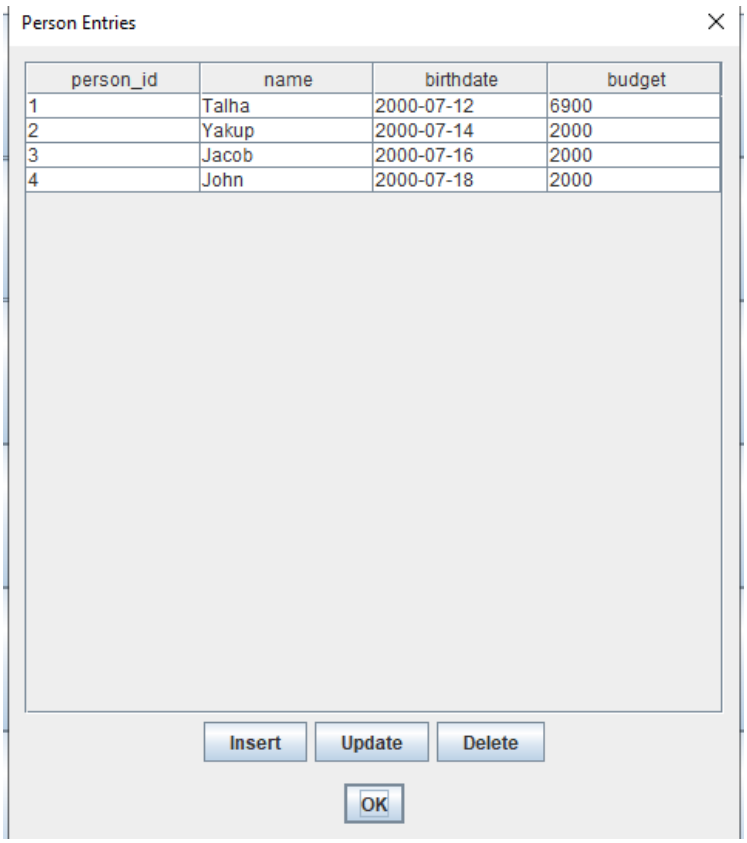
Person rents a car :
Look for person budget



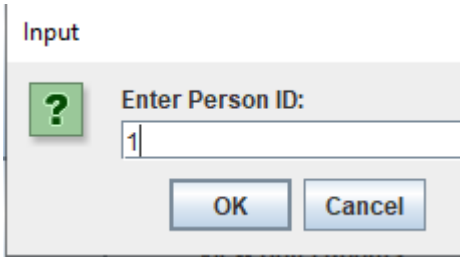


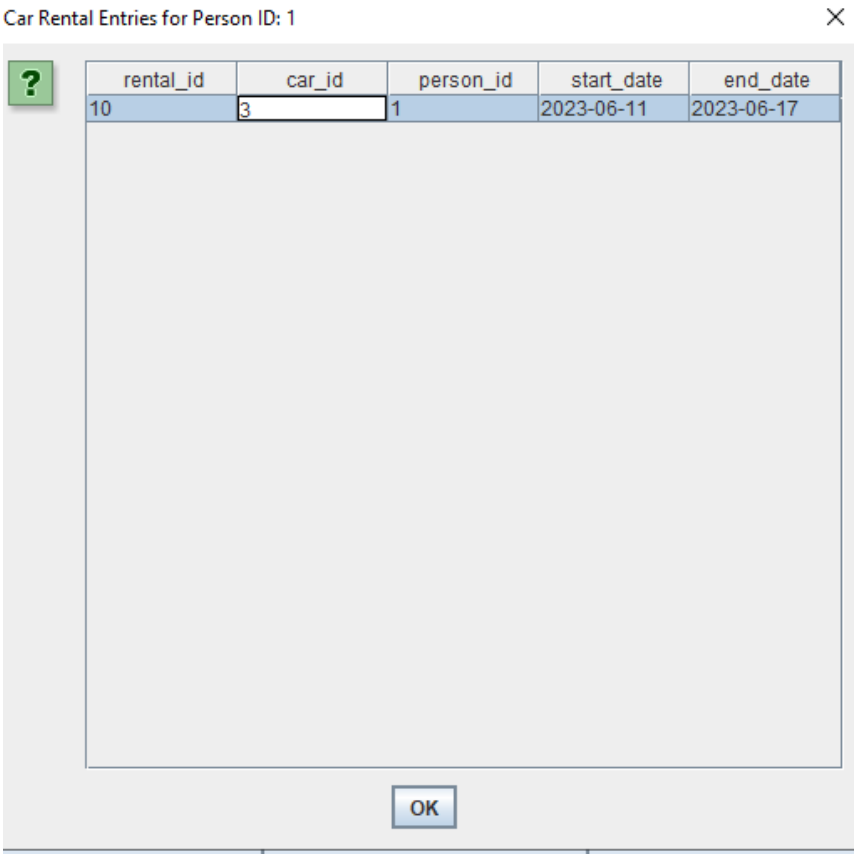


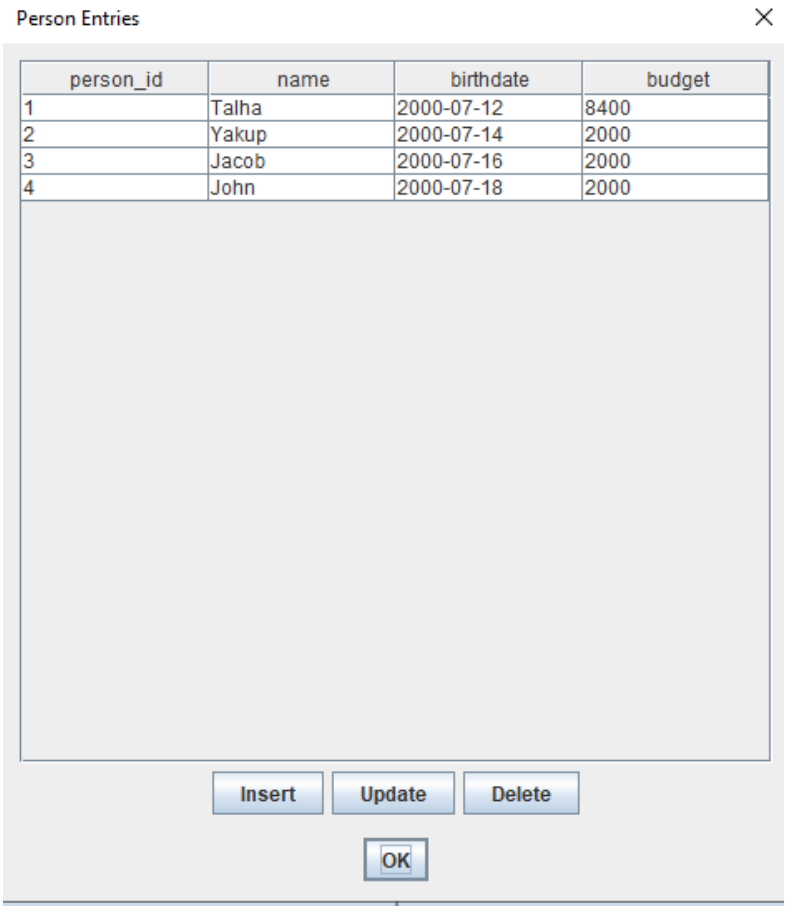
Budget is updated on trigger



lets cancel the car rental so that budget is updated again with a trigger



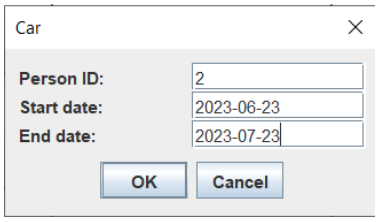


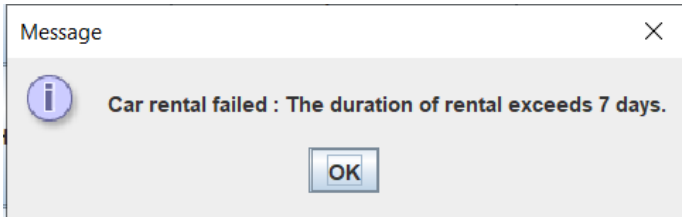


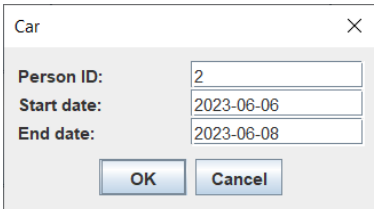
Can’t rent a car more than one week and check budget is enough

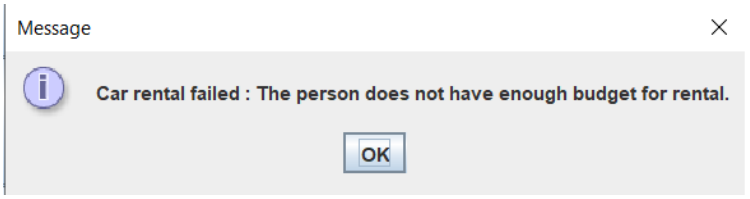
```
CREATE DEFINER=`root`@`localhost` TRIGGER `car_rental_BEFORE_INSERT` BEFORE INSERT ON `car_rental` FOR EACH ROW BEGIN
    DECLARE num_days INT;
    DECLARE total_cost INT;
    DECLARE person_budget INT;

    SET num_days = DATEDIFF(NEW.end_date, NEW.start_date);

    SELECT daily_price * num_days INTO total_cost
    FROM car
    WHERE car_id = NEW.car_id;

    SELECT budget INTO person_budget
    FROM person
    WHERE person_id = NEW.person_id;

    IF num_days > 7 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The duration of rental exceeds 7 days.';
    END IF;

    IF total_cost > person_budget THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The person does not have enough budget for rental.';
    END IF;
END
```

Try to rent a car



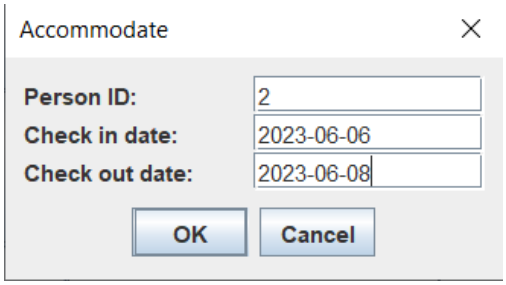


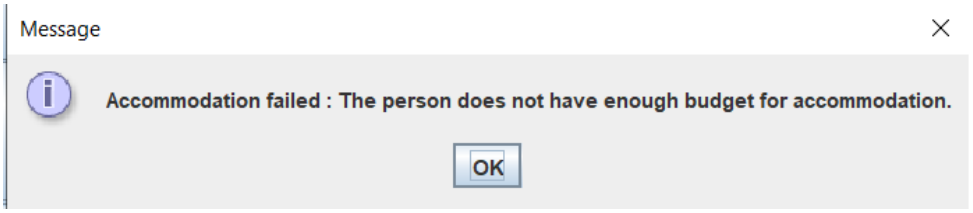


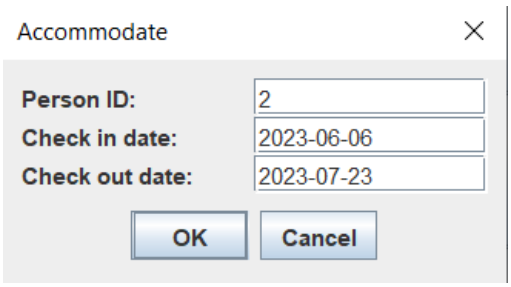


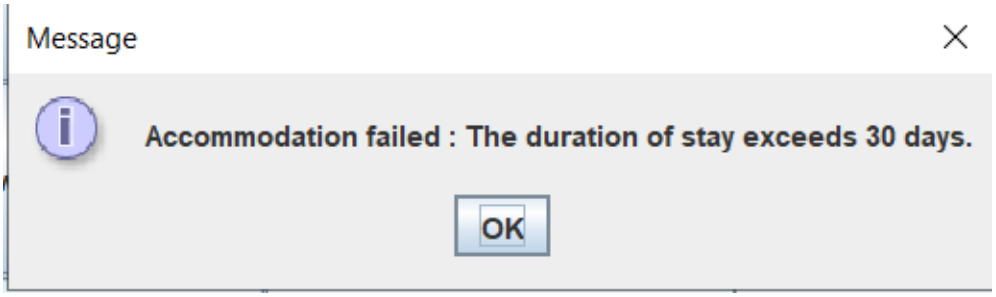
Cant accommodate on a hotel more than one month and check budget is enough

```
CREATE DEFINER=`root`@`localhost` TRIGGER `accommodation_BEFORE_INSERT` BEFORE INSERT ON
`accommodation` FOR EACH ROW BEGIN
    DECLARE num_days INT;
    DECLARE total_cost INT;
    DECLARE person_budget INT;

    SET num_days = DATEDIFF(NEW.check_out_date, NEW.check_in_date);

    SELECT daily_price * num_days INTO total_cost
    FROM room
    WHERE room_id = NEW.room_id AND hotel_id = NEW.hotel_id;

    SELECT budget INTO person_budget
    FROM person
    WHERE person_id = NEW.person_id;

    IF num_days > 30 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The duration of stay exceeds 30 days.';
    END IF;

    IF total_cost > person_budget THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The person does not have enough budget for accommodation.';
    END IF;
END
```



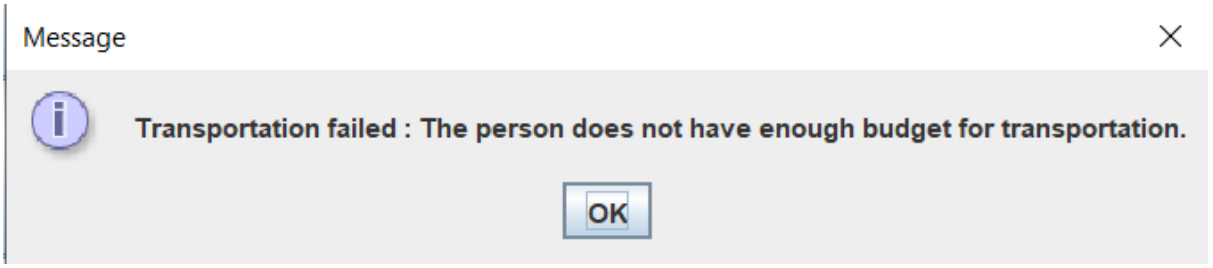






Check budget for transportation

```
CREATE DEFINER=`root`@`localhost` TRIGGER `transport_BEFORE_INSERT` BEFORE INSERT ON `transport` FOR EACH
ROW BEGIN
    DECLARE total_cost INT;
    DECLARE person_budget INT;
            DECLARE respective_id INT;
    DECLARE tr_type  VARCHAR(255);


    SELECT budget INTO person_budget
    FROM person
    WHERE person_id = NEW.person_id;

            SET tr_type = NEW.transport_type;
    SET respective_id = NEW.respective_id;

    IF tr_type = "flight" THEN
                        SELECT price INTO total_cost
        FROM flight
        WHERE respective_id = flight_id;

    END IF;

    IF tr_type = "bus" THEN
                        SELECT price INTO total_cost
        FROM bus
        WHERE respective_id = bus_id;
    END IF;


    IF total_cost > person_budget THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'The person does not have enough budget for transportation.';
    END IF;
END
```

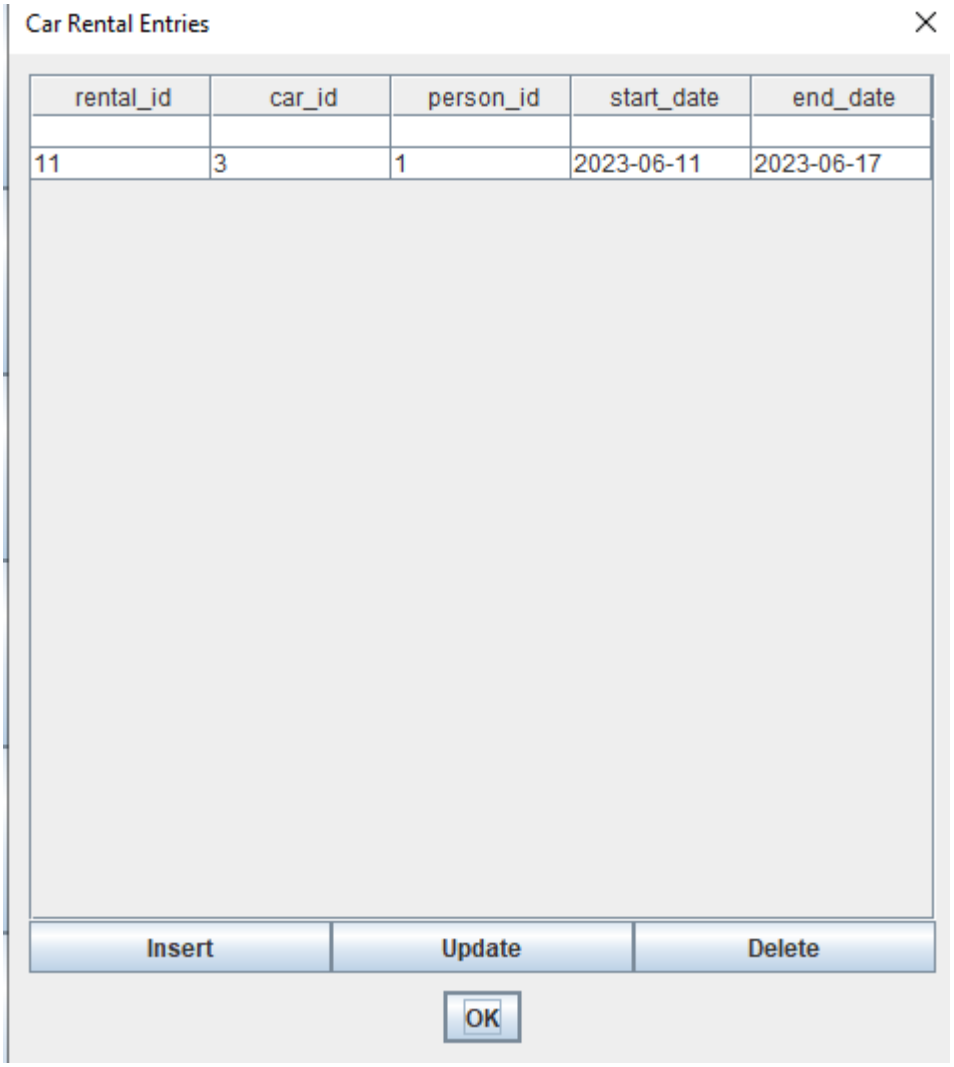


JOINS

RIGHT JOIN

While company is viewing car rentals :

```
String query = "SELECT DISTINCT car_rental.rental_id, car_rental.car_id, car_rental.person_id, car_rental.start_date, car_rental.end_date "
        "FROM car_rental " +
        "RIGHT JOIN car ON car.car_id = car_rental.car_id " +
        "WHERE car.company_id = " + company_id;
```



LEFT JOIN

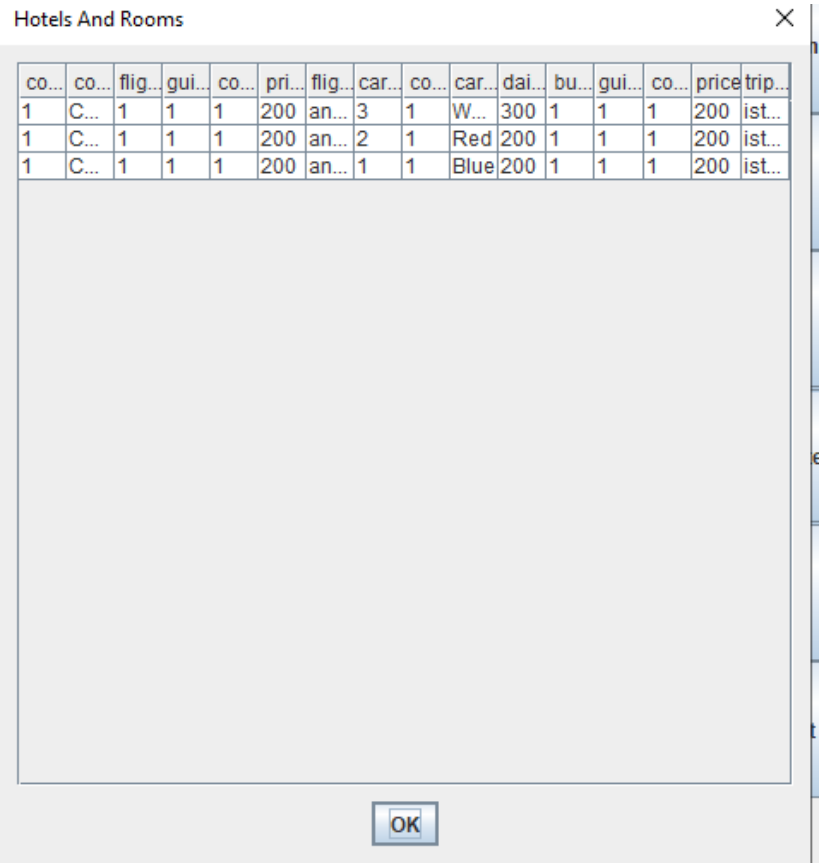
While company is viewing rooms of a hotel

```
String query = "SELECT room.room_id, room.hotel_id, room.daily_price, room.room_properties " +
        "FROM room " +
        "LEFT JOIN hotel ON room.hotel_id = hotel.hotel_id " +
        "WHERE room.hotel_id = "+hotelID;
```

FULL JOIN

My MySQL version does not support FULL JOIN keyword, I used this instead :
Person is viewing all hotels and all rooms

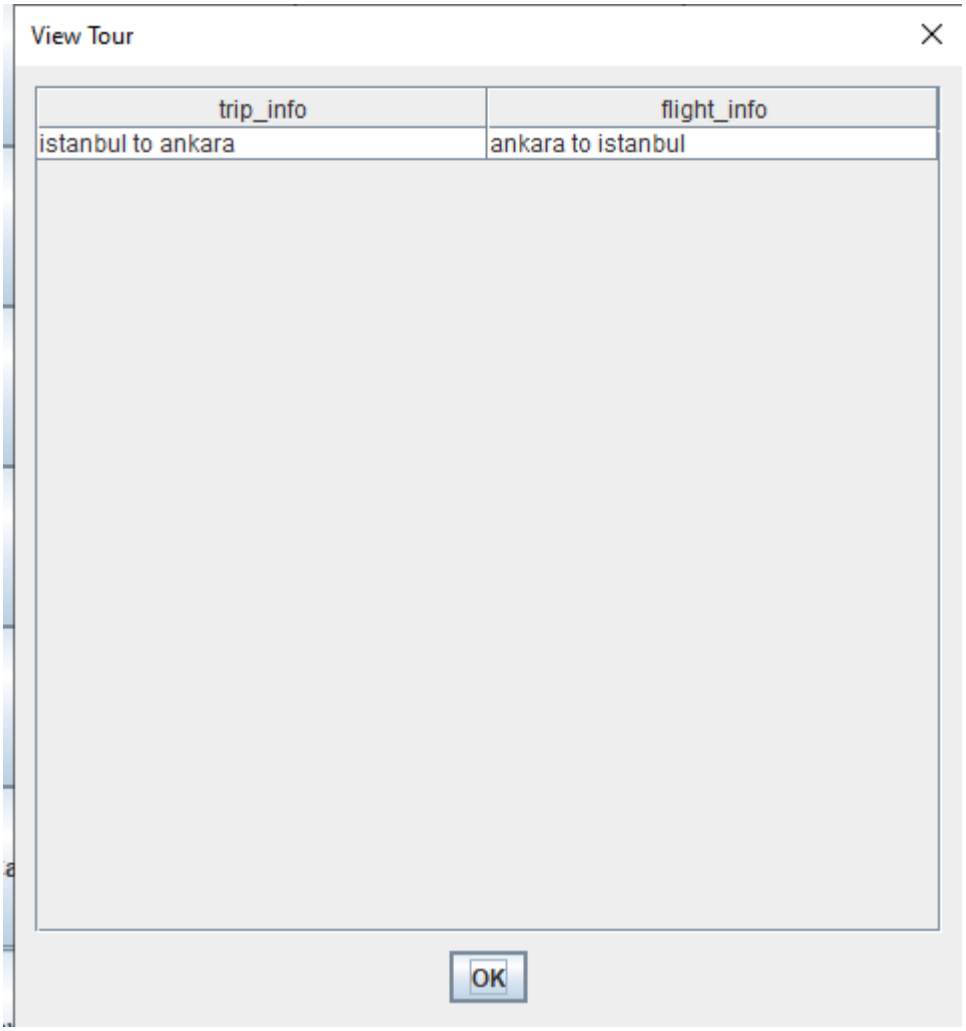
```
String query = "SELECT *\n" +
        "FROM company\n" +
        "LEFT JOIN flight ON company.company_id = flight.company_id\n" +
        "LEFT JOIN car ON company.company_id = car.company_id\n" +
        "LEFT JOIN bus ON company.company_id = bus.company_id\n" +
        "UNION\n" +
        "SELECT *\n" +
        "FROM company\n" +
        "RIGHT JOIN flight ON company.company_id = flight.company_id\n" +
        "RIGHT JOIN car ON company.company_id = car.company_id\n" +
        "RIGHT JOIN bus ON company.company_id = bus.company_id\n" +
        "WHERE company.company_id =" + companyID ;
```



INNER JOIN

When person wants to see the companies flight and bus infos together

```
try {
    // Prompt the user to enter the company ID
    String companyID = JOptionPane.showInputDialog( parentComponent: this,
            JOptionPane.QUESTION_MESSAGE);

    String query = "SELECT bus.trip_info,flight.flight_info\n" +
            "FROM bus\n" +
            "INNER JOIN flight ON bus.company_id = flight.company_id;";
```

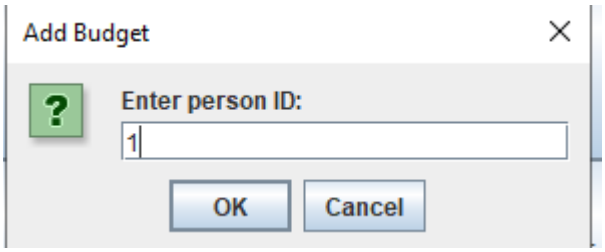


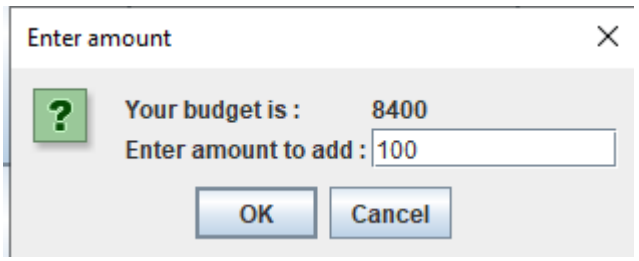
ATOMIC TRANSACTIONS

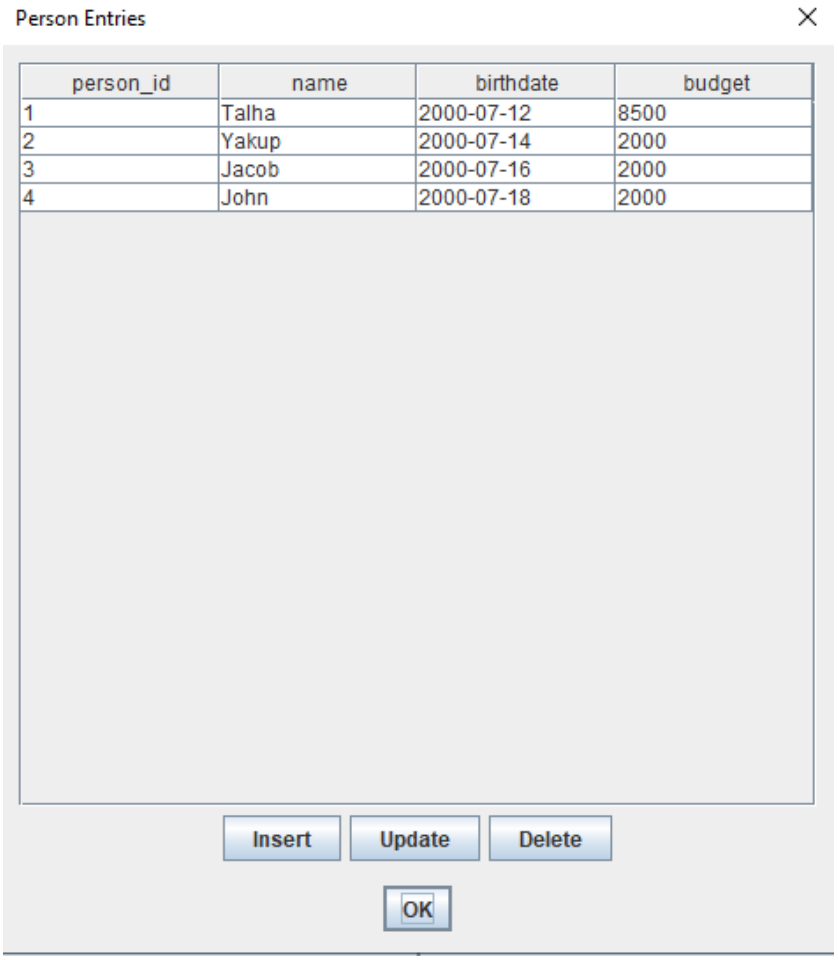
While person adds a budget

DDL:

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `para_transfer`(IN personid INT,IN amount INT)
2  BEGIN
3  START TRANSACTION ;
4  UPDATE person SET budget = budget + amount WHERE person_id = personid;
5  COMMIT;
6  END
```

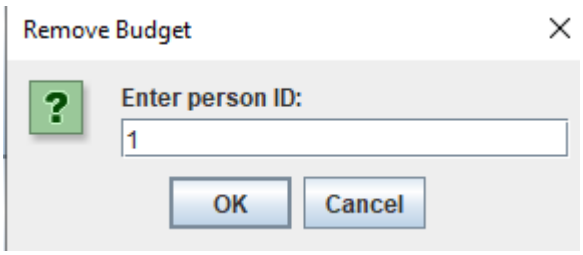


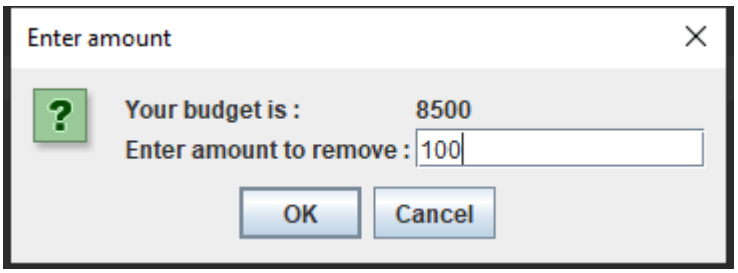


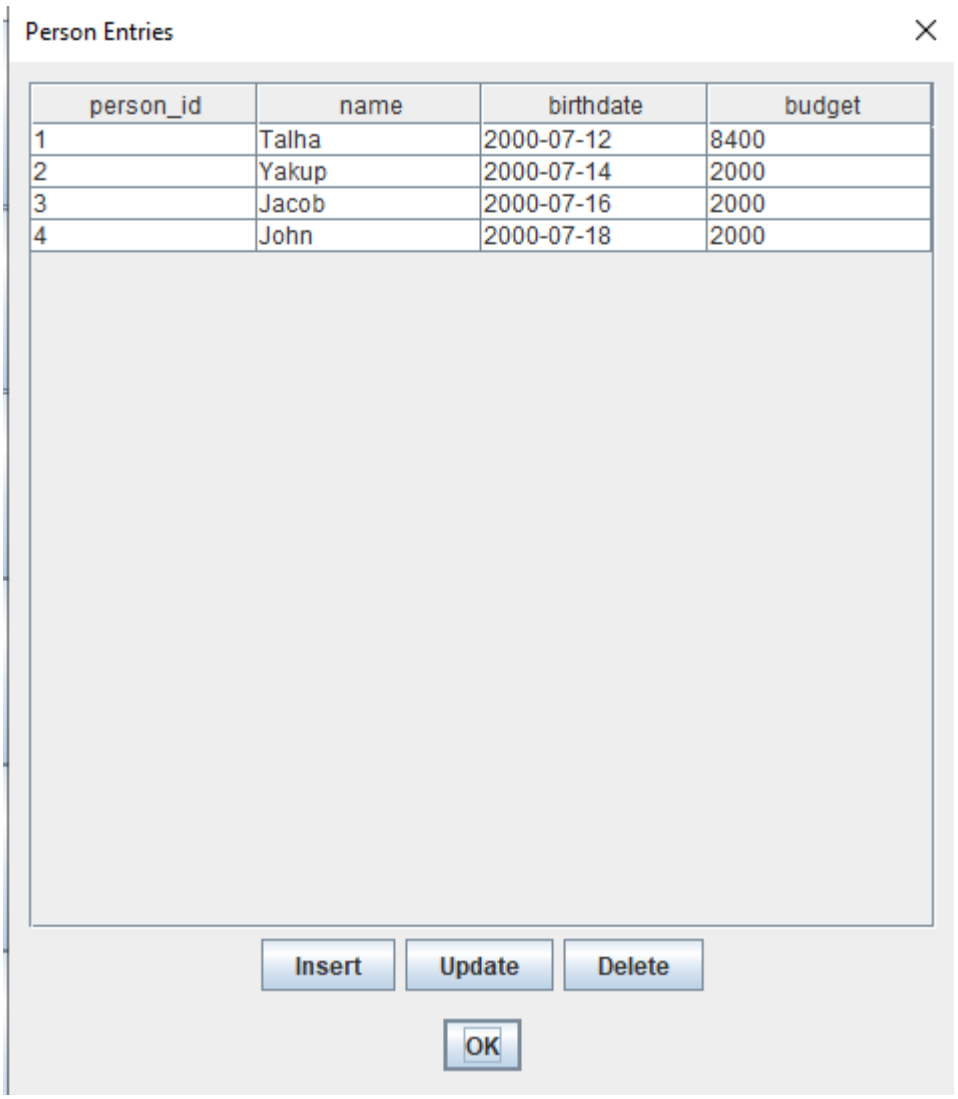


While person removes a budget

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `para_cek`(IN personid INT,IN amount INT)
BEGIN
START TRANSACTION ;
UPDATE person SET budget = budget - amount WHERE person_id = personid;
COMMIT;
END
```

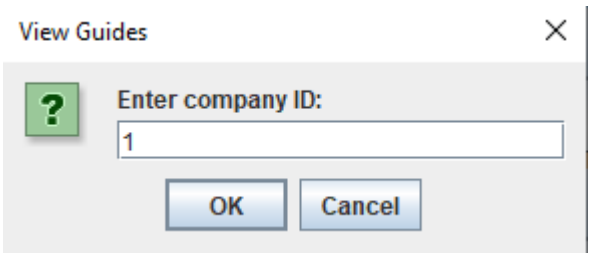




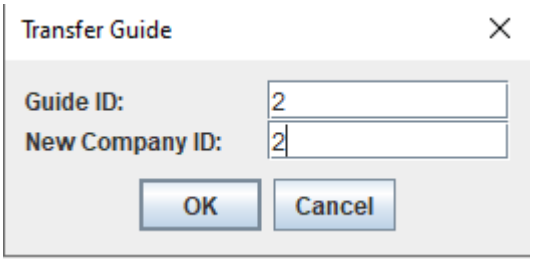


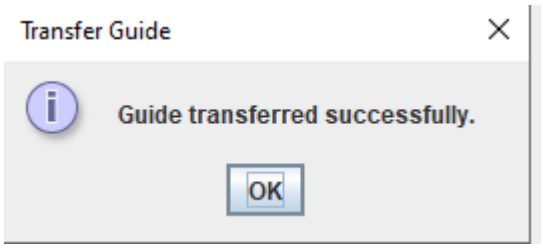
While company transfers a guide

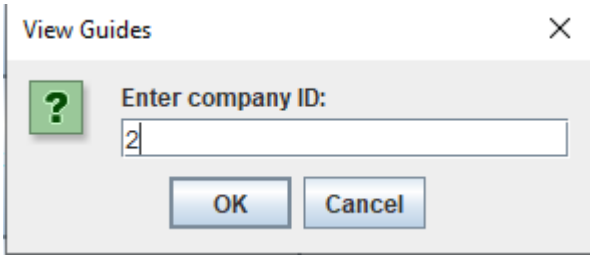
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `transfer_guide`(IN guideID INT,IN newCompanyID INT)
BEGIN
  -- Start the transaction
  START TRANSACTION;

  -- Update the guide's company_id to the newCompanyID
  UPDATE guide
  SET company_id = newCompanyID
  WHERE guide_id = guideID;

  -- Check if the update was successful
  IF ROW_COUNT() = 0 THEN
    -- Rollback the transaction if the guide ID was not found
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Guide ID not found.';
  ELSE
    -- Commit the transaction
    COMMIT;
  END IF;
END
```

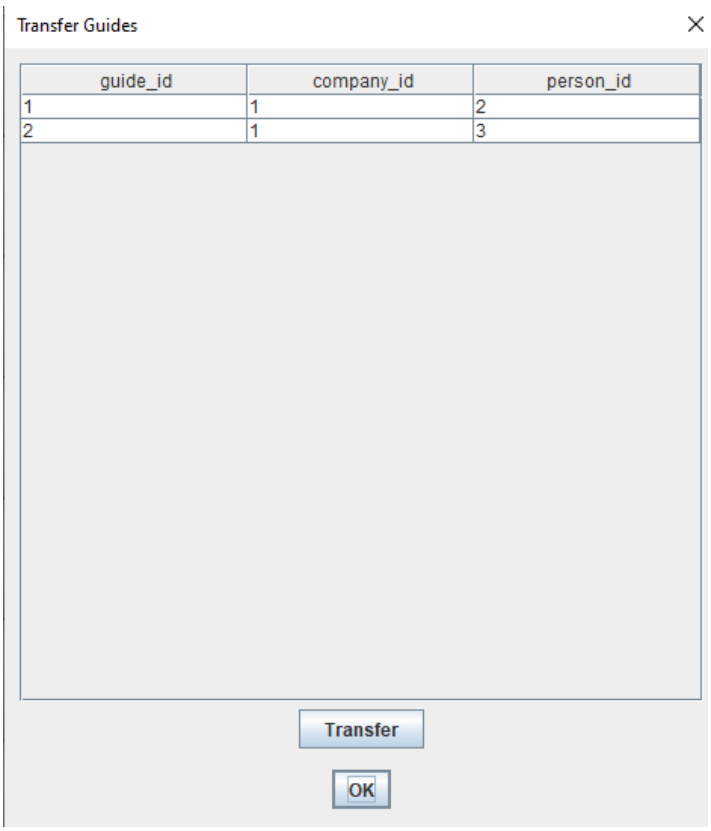


View and transfer the guide.

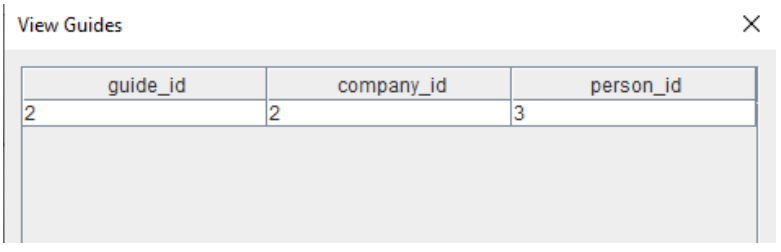






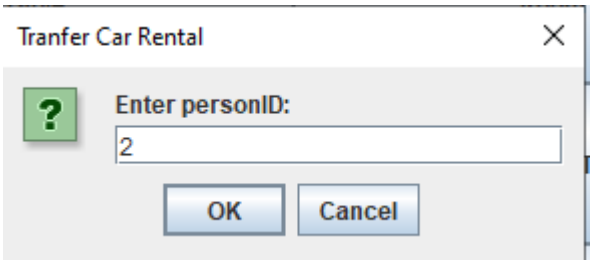


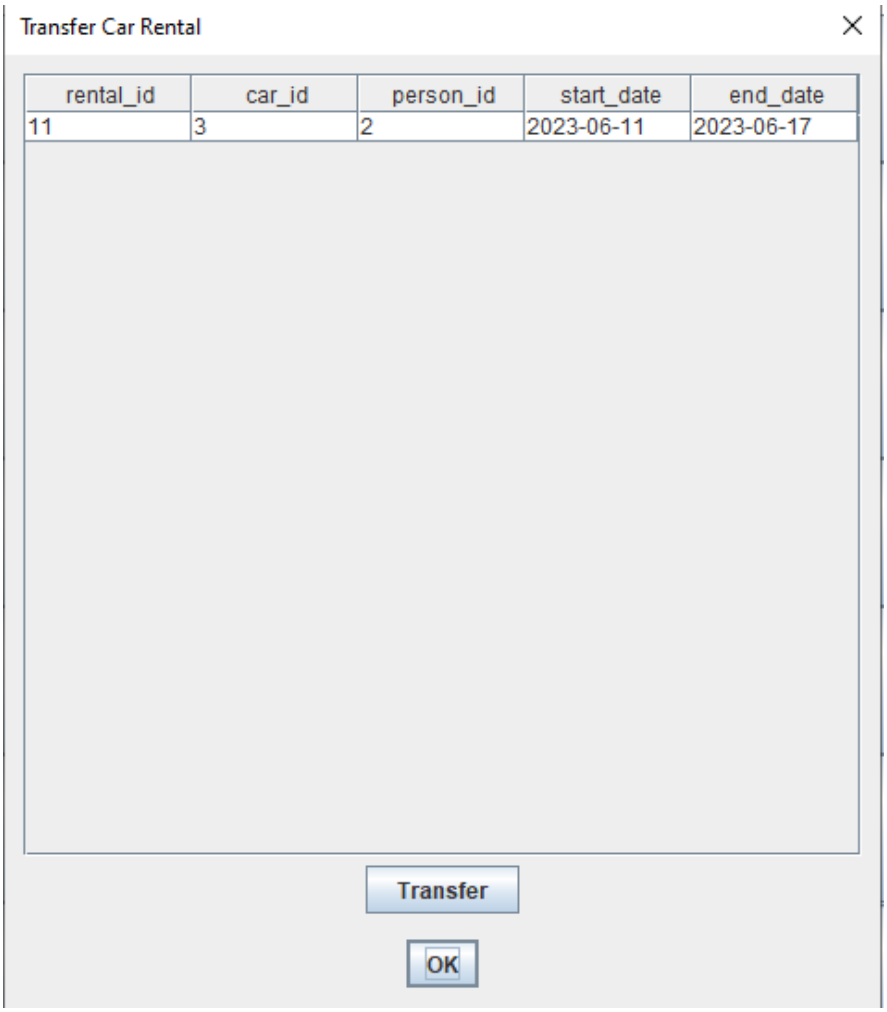
After transferring :



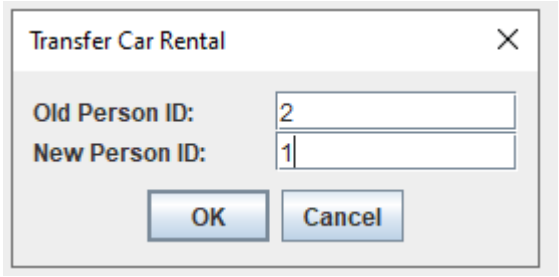
While company transfers a car_rental

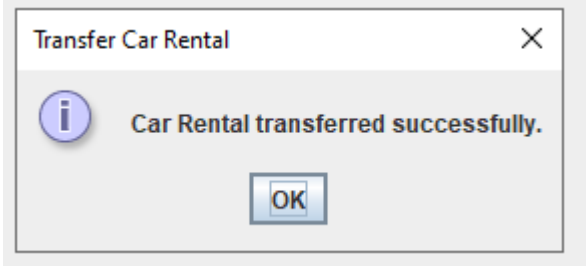
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `transfer_car_rental`(IN oldPersonID INT, IN newPersonID INT)
BEGIN
  DECLARE oldPersonBudget DECIMAL(10, 2);
  DECLARE newPersonBudget DECIMAL(10, 2);
  DECLARE rentalID INT;
  DECLARE dailyPrice INT;
  DECLARE startDate DATE;
  DECLARE endDate DATE;
  DECLARE carId INT;
  DECLARE rentalCost INT;

  -- Start the transaction
  START TRANSACTION;

  -- Check if the old person exists and get their budget
  SELECT budget INTO oldPersonBudget FROM person WHERE person_id = oldPersonID;
  IF oldPersonBudget IS NULL THEN
    -- Rollback the transaction if the old person does not exist
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Old person not found.';
  END IF;

  -- Check if the new person exists and get their budget
  SELECT budget INTO newPersonBudget FROM person WHERE person_id = newPersonID;
  IF newPersonBudget IS NULL THEN
    -- Rollback the transaction if the new person does not exist
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'New person not found.';
  END IF;

  SELECT rental_id,start_date,end_date,car_id INTO rentalID,startDate,endDate,carId FROM car_rental WHERE person_id = oldPersonID;
  IF rentalID IS NULL THEN
    -- Rollback the transaction if the car rental does not exist for the old person
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Car rental not found for the old person.';
  END IF;

  SELECT daily_price INTO dailyPrice FROM car WHERE car_id = carId;
  SET rentalCost = (dailyPrice * DATEDIFF(startDate, endDate));
  IF newPersonBudget < rentalCost THEN
    -- Rollback the transaction if the new person's budget is insufficient
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient budget for the transfer.';
  END IF;

  -- Update the old person's budget by deducting the rental cost
  UPDATE person SET budget = budget - rentalCost WHERE person_id = oldPersonID;

  -- Update the car_rental record with the new person's ID
  UPDATE car_rental SET person_id = newPersonID WHERE rental_id = rentalID;

  -- Check if the update was successful
  IF ROW_COUNT() = 0 THEN
    -- Rollback the transaction if the rental ID was not found
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Rental ID not found.';
  ELSE
    -- Commit the transaction
    COMMIT;
  END IF;
```





Transfer:





Updated table :